به نام او

دانشگاه صنعتی خواجه نصیرالدین طوسی
گروه مهندسی برق

عنوان :

تمرین مینی پروژه سوم یادگیری ماشین

دانشجو:

آرمان مرزبان

۴۰۲۰۰۹۱۶

استاد:

دکتر علیاری

ماه و سال

خرداد ماه ۱۴۰۳

فهرست مطالب

فهرست شکل‌ها

بخش اول

سوال اول

۱-۱- هدف از این سوال آزمایش الگوریتم SVM در نمونه های مختلف روی دیتاست معروف گل زنبق است. مراحل زیر را یک به یک انجام دهید و موارد خواسته شده در گزارش خود به همراه کدها ارسال کنید.

آ. در مرحلهٔ اول دیتاست را فراخوانی کنید و اطلاعاتی نظیر ابعاد، تعداد نمونه ها، میانگین، واریانس و همبستگی ویژگی ها را به دست آورید و نمونه های دیتاست را به تصویر بکشید (مثلا با استفاده از t SNE .) سپس، با توجه به اطلاعات عددی، آماری و بصری بدست آمده، تحلیل کنید که آیا کاهش ابعاد می تواند در این دیتاست قابل استفاده باشد یا خیر.

حل:

اطلاعت دیتاست گل زنبق:

معرفی:

دیتاست گل زنبق (Iris flower data set) یا مجموعه داده زنبق فیشر، یک مجموعه داده چند متغیره است که توسط رانلد فیشر، آماردان و زیست‌شناس بریتانیایی در سال ۱۹۳۶ معرفی شد .این مجموعه داده به دلیل سادگی و کاربرد گسترده در آموزش الگوریتم‌های یادگیری ماشین، به یکی از محبوب‌ترین مجموعه داده‌ها در علوم داده تبدیل شده است.

محتوا:

این مجموعه داده شامل ۱۵۰ نمونه از گل زنبق است که از سه گونه مختلف جمع‌آوری شده‌اند Iris setosa ، Iris virginicaو .Iris versicolor برای هر نمونه، چهار ویژگی اندازه‌گیری شده است:

- طول کاسبرگ(Sepal length)
- عرض کاسبرگ(Sepal width)
- طول گلبرگ(Petal length)
- عرض گلبرگ(Petal width)

شکل هر سه گل به صورت آورده شده در شکل ۱ است:

شکل (۱): سه گل محتلف زنبق

ابتدا با دستور زیر داده ها را فراخولنی می کنیم:

```
data = pd.read_csv('iris.csv')
```

با دستور زیر مشخصات و head های جدولی داده را نمایش می دهیم:

```
#looking at the first 5 values of the dataset
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

نتیجه بدست آمده نمایش ۵ نمونه از ویژگی ها و خروجی است که نوع گل و در ستون آخر داده ها قرار دارد.

با استفاده از دستور زیر تعداد نمونه و ویژگی ها و ورودی- خروجی استخراج شده است:

```
X = data[['sepal_length', 'sepal_width', 'petal_length',
'petal_width']]
y = data[['variety']]
print("X.shape :", X.shape)
print(" Target.shape:",y.shape)
```

```
X.shape : (150, 4)
 Target.shape: (150, 1)
```

نتیجه همانطور که بالاتر اشـاره شـد ۱۵۰ نمونه با چهار ویژگی و ۱۵۰ نمونه در ۳ کلاس اسـت که به عنوان خروجی تعریف شده است.

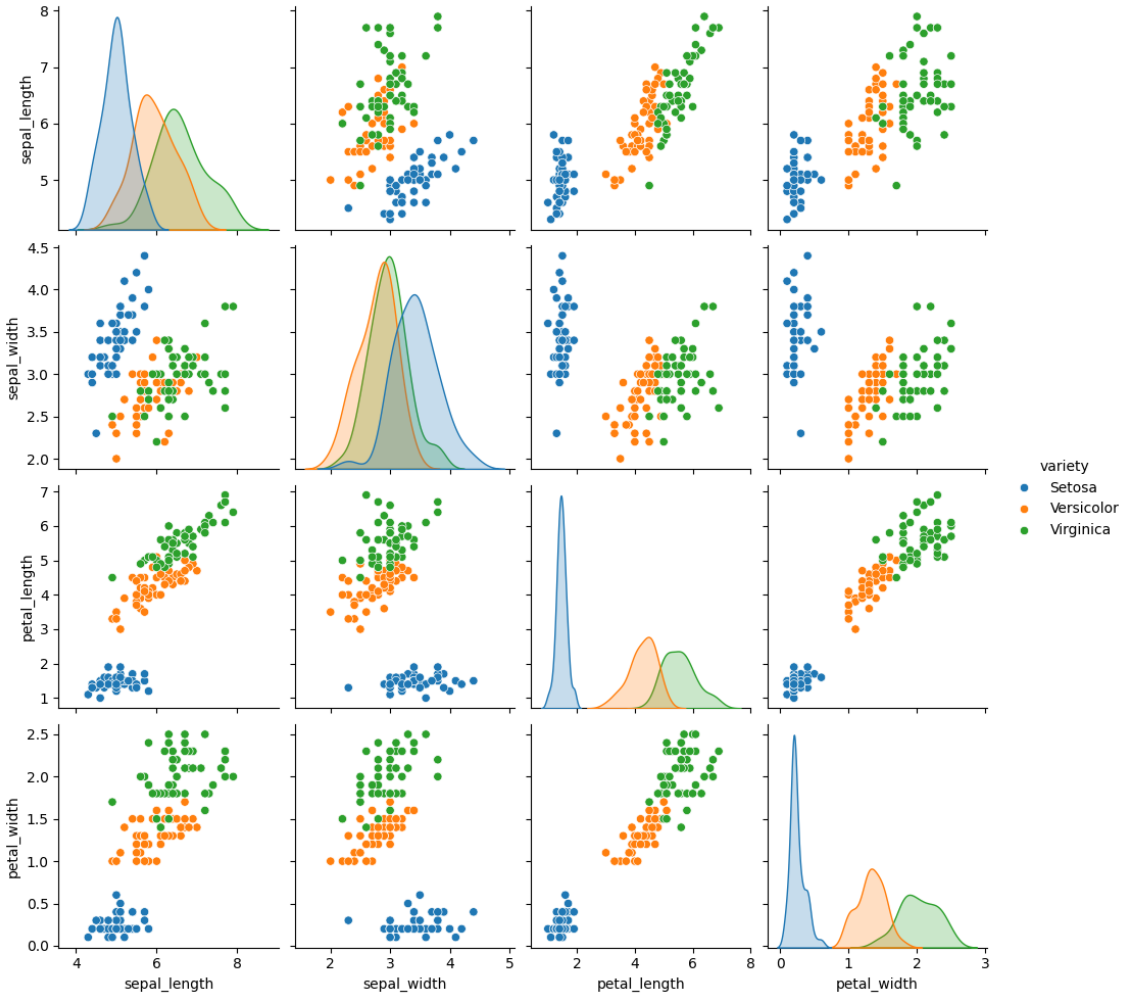
با استفاده از دستور زیر اطلاعات دیتاست را استخراج می نماییم:

```
data.info()
```

```
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   variety       150 non-null    object
```

مشاهده می شود که در هیچ یک از ستون ها داده nall وجود ندارد.

با اسـتفاده از دسـتور زیر موقعیت جفتی داده ها با در نظر گرفتن بردار variety به عنوان خروجی به صـورت شکل زیر حاصل شده است:



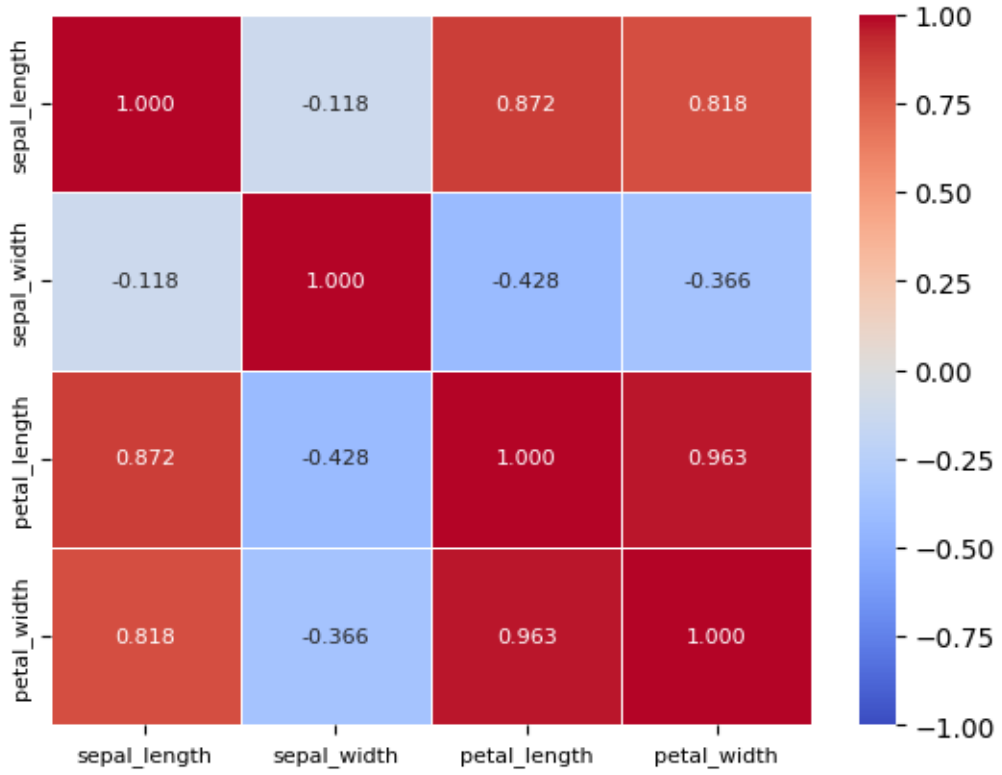
شکل ۲: موقعیت جفتی داده ها

طبق موقعیت نوع گلها از روی شـــکل مشـــاهده می شـــود گل setosa با دو گل دیگر تقریبا به راحتی قابل تفکیک اســت و میزان چالش تفکیک با توجه به شــکلها در دو نمونه گل verginica و versicolor اســت. نمودارهای روی قطر اصلی نیز میزان توزیع **طول کاسبرگ، عرض کاسبرگ، طول گلبرگ و عرض گلبرگ** را به صورت جداگانه را نشان می دهند و ملاحظه می شود که توزیع آنها تقریبا نرمال است.

با استفاده از قطعه کد زیر هیت مپ یا همبستگی داده قابل استخراج است:

```
# # Calculating correlation matrix
corr_matrix = X.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm',
linewidths=0.5, annot_kws={"size": 8}, fmt='.3f',
yticklabels=corr_matrix.columns, vmin=-1, vmax=1) #
# Plot correlation results
# Adjust font size of annotations
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.show()
```

که در این حالت ماتریس همبستگی به صورت زیر بدست آمده است:



شکل ۳: ماتریس همبستگی داده های گل زنبق

باتوجه به مشخصات آماری بدست آمده و خصوصا ماتریس همبستگی نشان داده شده در شکل ۳، مشاهده می شود میزان همبستگی یا کوواریانس داده ها بسیار زیاد است به عنوان مثال داده های petal_length و داده های petal_width ۹۶.۳ درصد به یکدیگر وابسته اند و و این ویژگی اطلاعالت جدیدی به ما نخواهد داد لذا می توان با کاهش بعد از محاسبات اضافی برای ماشین جلوگیری نمود و به داده های بهتر و نتایج بهتر و مطلوب تری دست یافت.

ب. با استفاده از الگوریتمSVM ، با هستهٔ خطی، داده ها را طبقه بندی کنید و ماتریس درهم ریختگی آن را بدست آورید و مرزهای تصمیم گیری را در فضای دوبعدی (کاهش بعد از طریق یکی از روش های آموخته شده با ذکر دلیل) ترسیم کنید.

حل:

ابتدا با توجه به توضیحات شکل ۳ مقدار بعد را از ۴ به ۲ توسط قطعه کد زیر کاهش می دهیم:

```
from sklearn.manifold import TSNE
label_map = {"Virginica": 1, "Versicolor": 2, "Setosa": 3} #
Replace label to 1, 2, 3
y["variety"] = y["variety"].replace(label_map)
tsne = TSNE(n_components=2)
X_h = tsne.fit_transform(X)
```

همچنین در این کد جهت کار راحت تر بر روی داده ها برچسب گل ها Virginica، Verdicolor و Setosa به ترتیب با برچسب های عددی ۱، ۲ و ۳ جایگزین شده است. در ادامه به استفاده از مدل ماشین SVM با هسته خطی و تنظیم فراپارامترهای آن به صورت قطعه کد زیر:

```
model_Lin = LinearSVC(loss='hinge', C=0.001,tol=0.0001,
max_iter=2000)
model_Lin.fit(X_train, y_train)
accuracy_svm = model_Lin.score(X_test ,y_test )
print("test_accuracy:", accuracy_svm)
```

مدل را در ۲۰۰۰ گام آموزش می دهیم. در این حالت میزان بازدهی مدل بر روی داده های آزمون به میزان ۹۳.۳۳ درصد گزارش شده است. شایان ذکر است که پارامترهای C ، tol و max_iter تاثیر به سزایی در دقت مدل دارند. به عنوان چند نمونه که با آن در هنگام تنظیم فراپارامترها برخورد نمودم مواردی است که بازگویی می گردد. در ابتدا مقدار C=1,tol=0.01, max_iter=800 در نظر گرفته شد و خروجی مدل با دقت ۹۰ درصد به صورت زیر گزراش گردید:

```
test_accuracy: 0.9
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

مشاهده می شود با وجود دقت ن۹۰درصدی ولی در گزارش ذکر شده است که همگرایی صورت نگرفته است. اینبار مقدار گام ها را زیاد و به ۱۵۰۰ تغییر داده و نتیجه به صورت زیر گزارش گردید:

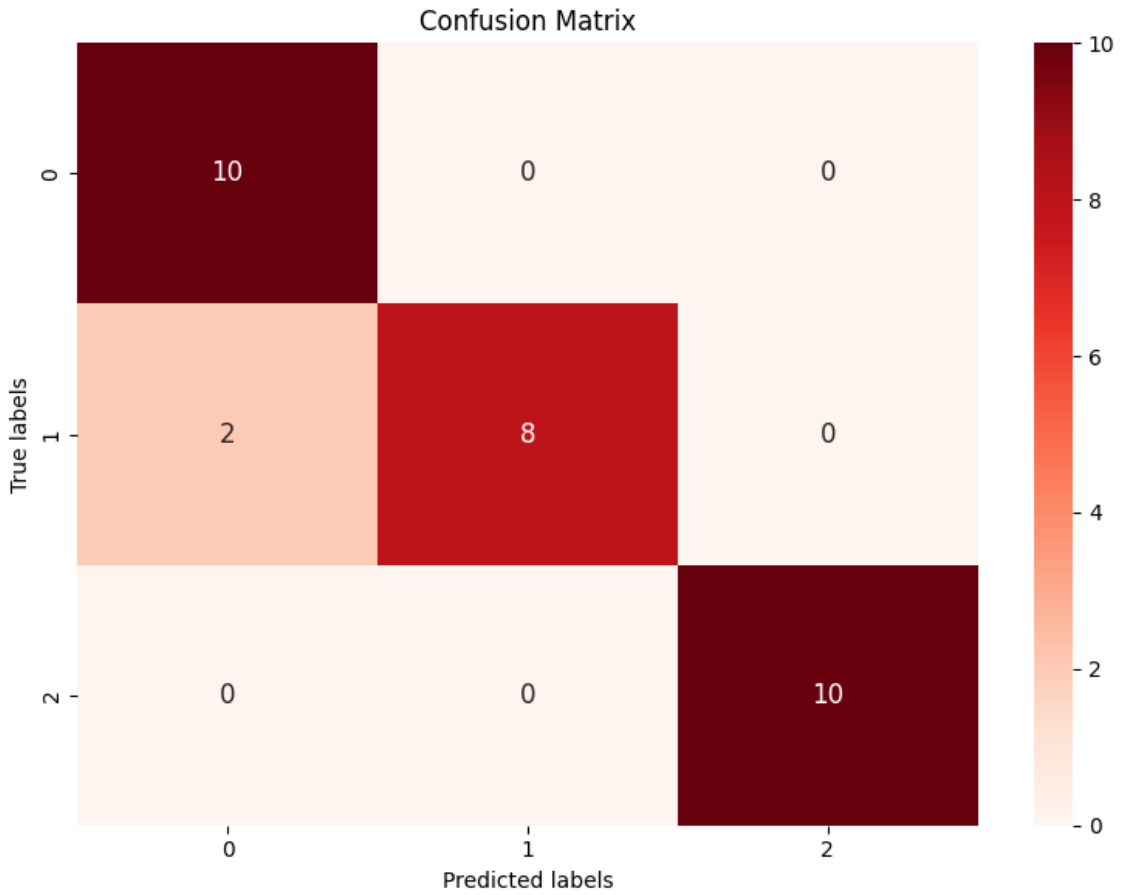
```
test_accuracy: 0.9333333333333333
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

مشاهده می شود با وجود ۳ درصد بیشتر شدن دقت اما همچنان پارامترها به صورت کامل همگرا نشده اند. با تنظیم فراپارامترها به صورت C=0.001,tol=0.0001, max_iter=2000 ,نتیجه آموزش مدل به صورت زیر گزارش گردید:

```
test_accuracy: 0.9333333333333333
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

مشـــاهده می شـــود با کاهش دادن مقدار پارامترهای C و tol و از طرفی افزایش تعداد گام ها به ۲۰۰۰، همگرایی مدل و پارامترها با دقت ۹۳.۳۳ درصدی بر روی داده های آزمون صورت گرفته است. ماتریس درهم ریختگی با استفاده از قطعه کد زیر به صورت شکل ۴ بدست آمده است:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
# Making predictions on the test set
 # Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred_Lin)

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='g', cmap='Reds',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for
matplotlib 3.1.1 and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
# print(classification_report(y_test, y_pred_mlp))
```



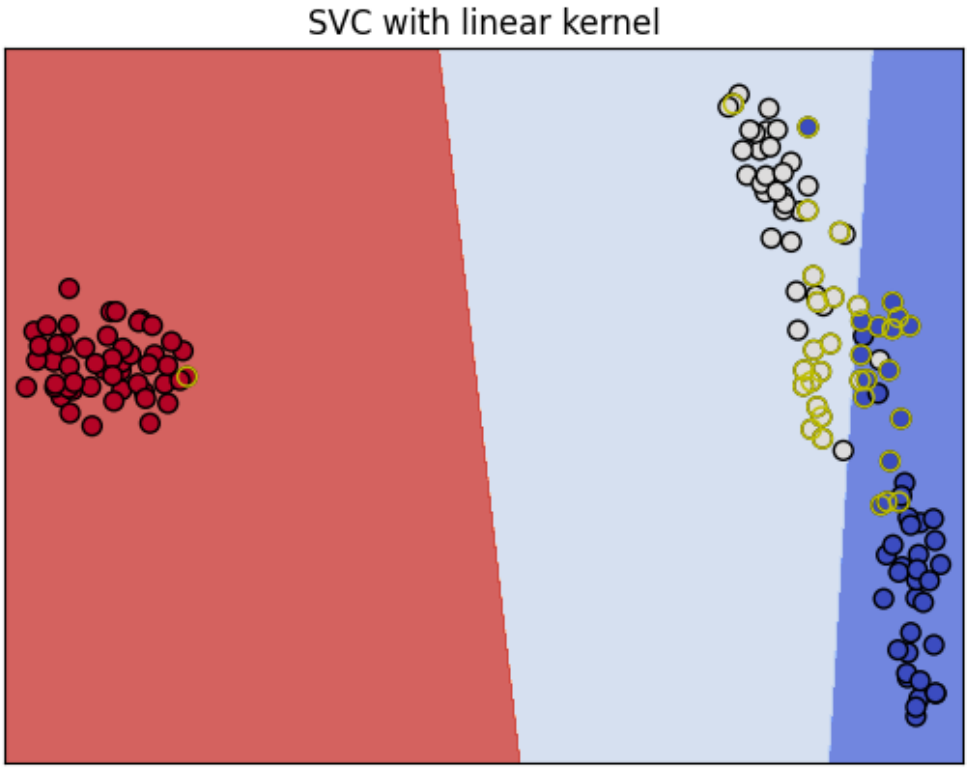
شکل ۴: ماتریس درهم ریختگی داده های گل زنبق با استفاده از کاهش بعد و روش SVM با هسته خطی

طبق شکل ۴ مشاهده می شود دو کلاس Virginica و Setosa به خوبی تفکیک شده اند و فقط دو نمونه از داده های کلاس Versicolor به اشتباهی توسط مدل در کلاس Virginica تفکیک شده است.

مرز تصمیم گیری نیز با استفاده از قطعه کد زیر به صورت شکل ۵ بدست آمده است:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
```

```
    return out

# Draw the decision region for the linear model
X0, X1 = X_h[:, 0], X_h[:, 1]
xx, yy = make_meshgrid(X0, X1)
fig, ax = plt.subplots()

# Titles for the plots
titles = ['SVC with linear kernel']
plt.subplots_adjust(wspace=1, hspace=1)
plot_contours(ax, model_Lin, xx, yy, cmap=plt.cm.coolwarm,
alpha=0.8)
ax.scatter(X0, X1, c=y["variety"], cmap=plt.cm.coolwarm, s=50,
edgecolors='k')
ax.scatter(
        model_Lin.support_vectors_[:, 0],
        model_Lin.support_vectors_[:, 1],
        s=50,
        facecolors="none",
        edgecolors="y",
    )
ax.set_xticks(())
ax.set_yticks(())
ax.set_title('SVC with linear kernel')

plt.show()
```



شکل ۵: ناحیه تصمیم گیری با استفاده از svm خطی برای داده های گل زنبق

در این شکل دایره هایی که با رنگ زرد مشخص شده اند همان S.V یا بردار پشتیبان هستند.

ج)

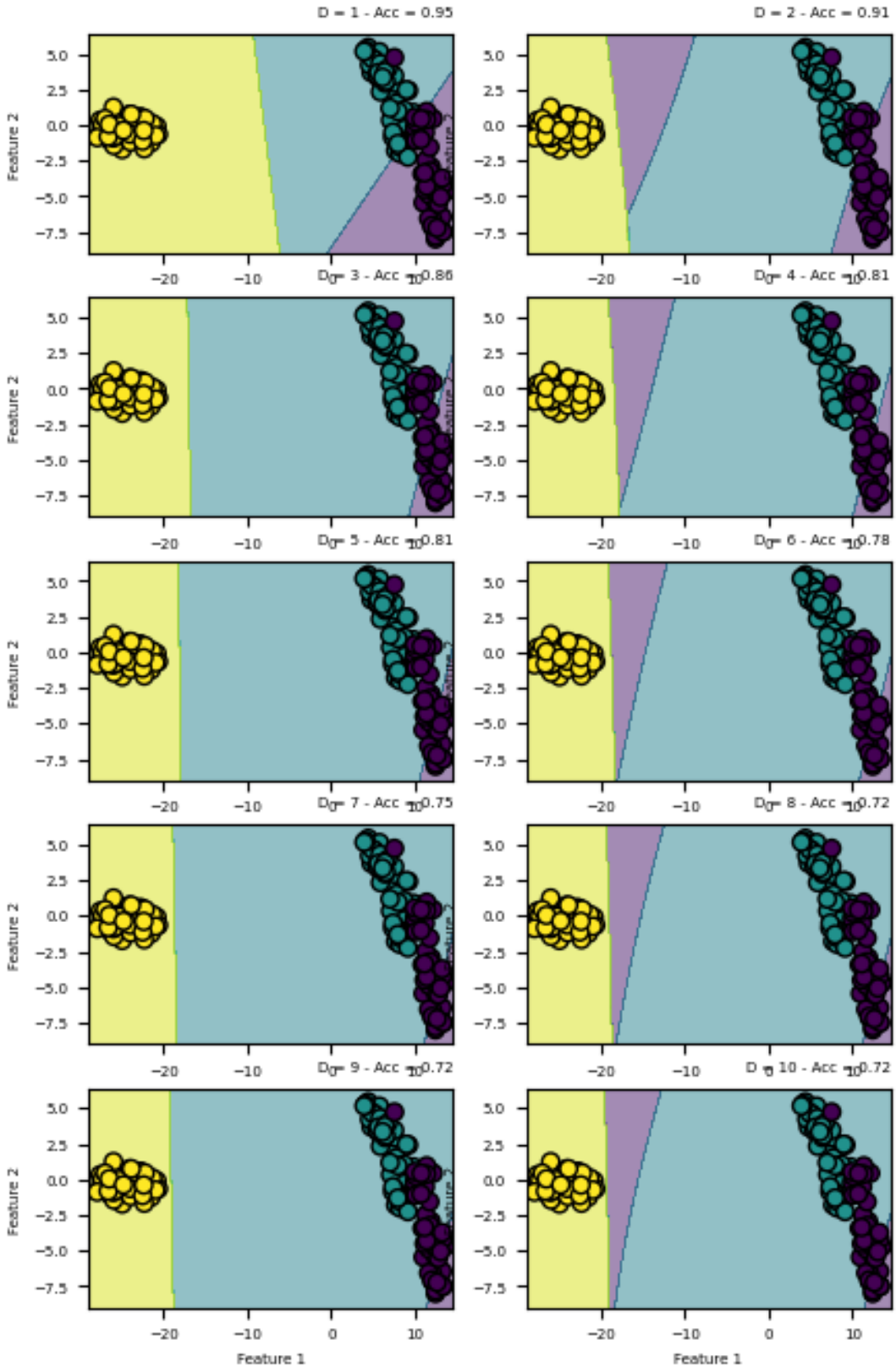
حل:

در این قسمت با استفاده از کرنل چند جمله ای و به ازای درجه های مختلف ۱ الی ۱۰ کار طبقه بندی داده ها را انجام می دهیم. برای اینکار قطعه کد زیر را نوشته ایم:

```
# Classification with polynomial kernel from one degree to ten
degree
from sklearn.metrics import accuracy_score

def plot_decision_boundaries_subplots(X, y):
    degrees = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    # degrees = [1, 2]
    models = [SVC(kernel='poly',C=0.1,tol=0.0001, degree=degree,
max_iter=2000) for degree in degrees]

    fig, axes = plt.subplots(5, 2, figsize=(5, 8))
    for ax in axes.flat:
        ax.tick_params(labelsize=2)
    axes = axes.flatten()

    for ax, model, degree in zip(axes, models, degrees):
        model.fit(X, y)
```

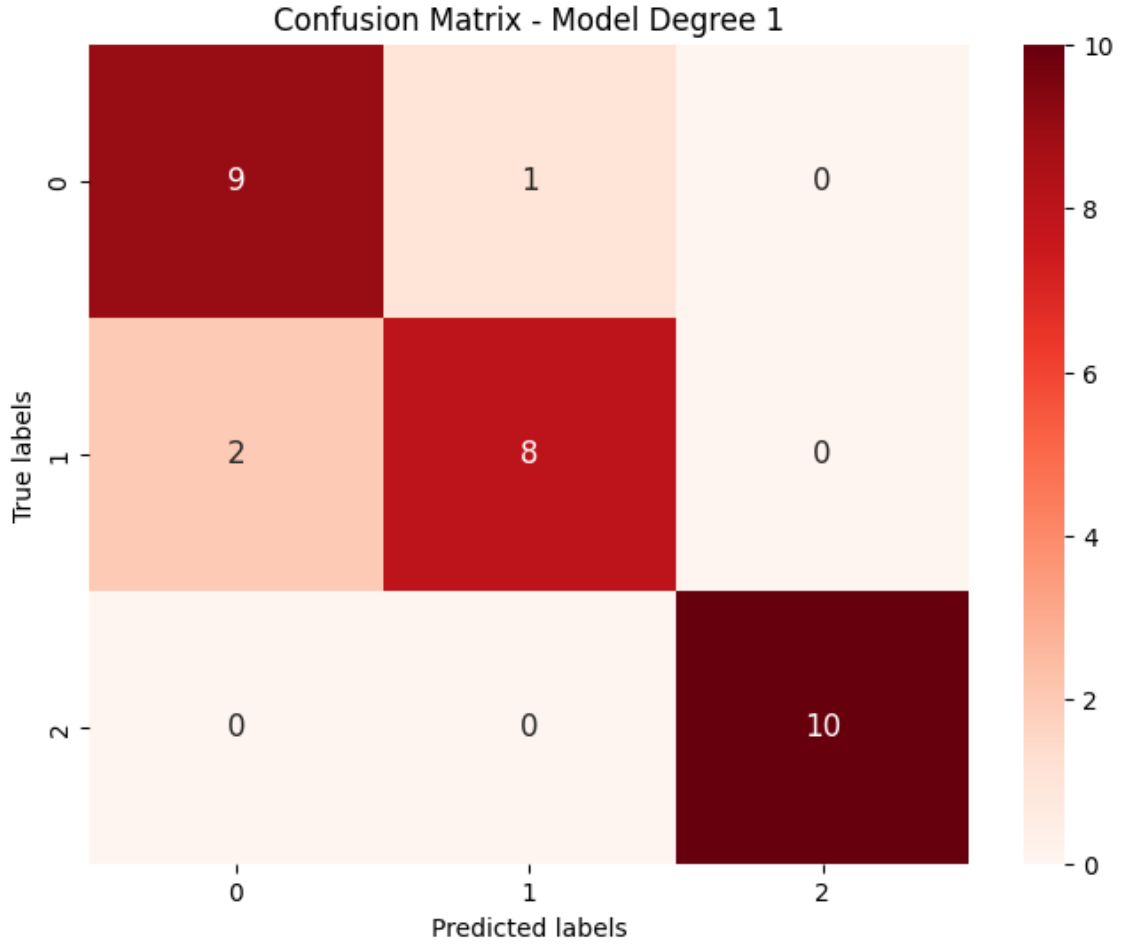
```
        y_pred = model.predict(X)
        accuracy = accuracy_score(y, y_pred)

        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                             np.arange(y_min, y_max, 0.01))
        Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)

        ax.contourf(xx, yy, Z, alpha=0.5)
        scatter = ax.scatter(X_h[:, 0], X_h[:, 1],
c=y['variety'], edgecolors='k')
        ax.set_title(f'D = {degree} - Acc = {accuracy:.2f}',
fontsize=5,loc ='right')
        ax.set_xlabel('Feature 1',fontsize=5)
        ax.set_ylabel('Feature 2',fontsize=5)

    plt.show()
plot_decision_boundaries_subplots(X_h, y)
```

نتیجه اجرای این کد به صورت شکل زیر بدست آمده است:

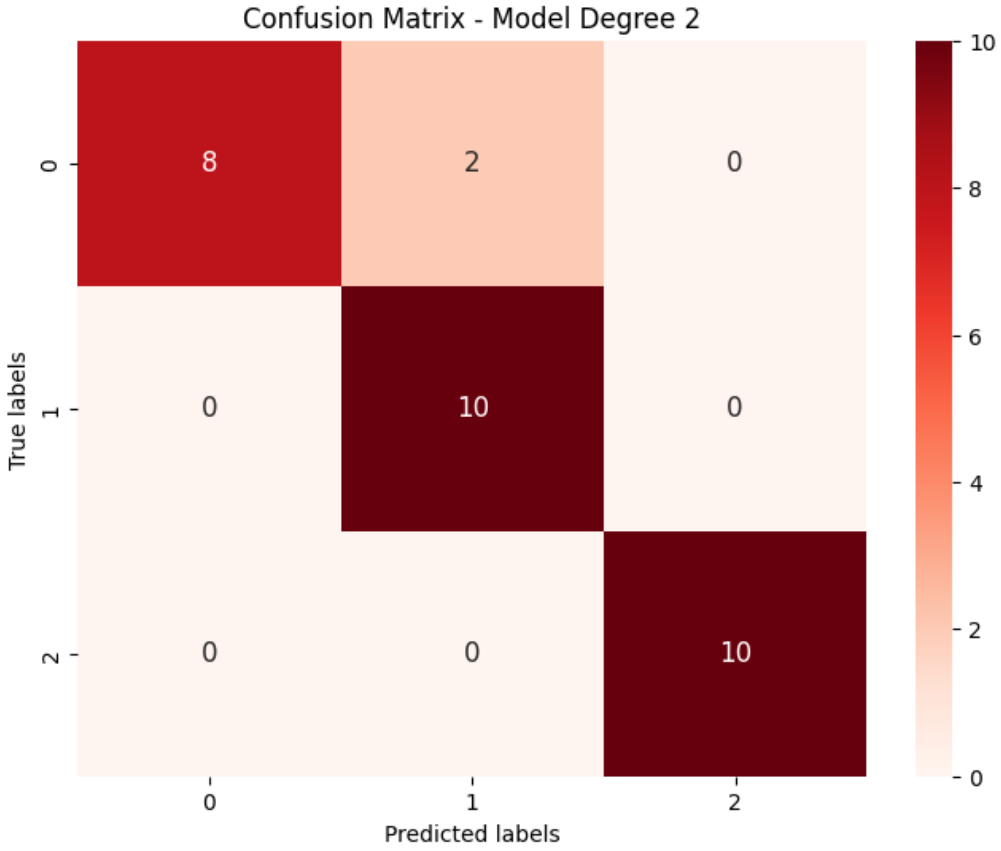


شکل ۶: ناحیه تصمیم به ازای کرنل چند جمله ای از درجه ۱ الی ۱۰(لینک آنلاین با کاهش بعد به روش TSNE)

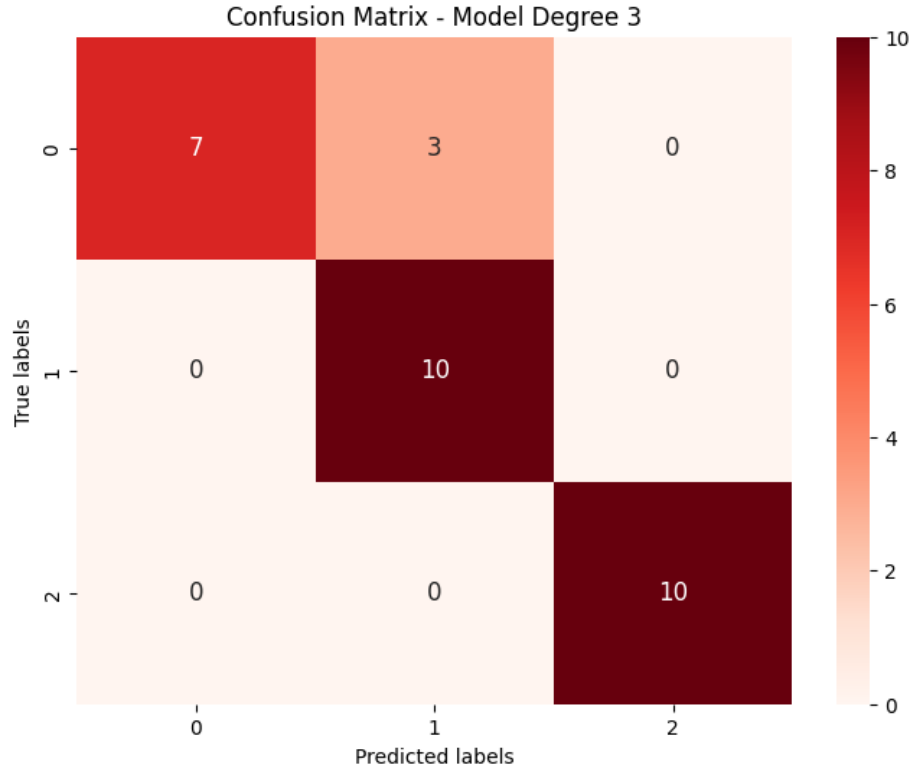
طبق شکل ملاحظه می شود بازدهی مدل به ازای درجه های ۱ و ۲ بسیار خوب و قابل قبول است اما وقتی مرتبه را بیشتر و مدل را پیچیده تر در نظر می گیریم، نه تنها دقت مدل بیشتر نشده است بلکه میزان آن به سرعت درحال کاهش یافتن است. بنابراین می توان نتیجه گرفت پیچیده کردن بی مورد باعث از دست دادن بازدهی خواهد شد و نباید مدل را زیاد پیچیده کرد چون در مدل های پیچیده عمل بیش برازش خیلی محتمل است و تاحد امکان نباید با انتخاب نامناسب فرا پارامترها سبب به وجود آمدن چنین پدیده ای شویم.همچنین با کاهش بعد داده به روش PCA نتایج قابل توجهی بدست آمد که طبق شکل بالا در روش کاهش بعد TSNE ناحیه های تصمیم به ازای درجه های چند جمله ای ۱ تا ۱۰ بیشتر به صورت یک خط است اما در روش PCA که همگرایی پارامترها در بعضی موارد و به ازای درجه های بالاتر رخ نمیداد و ناحیه های تصمیم به صورت های مختلف رسم گردید که نتایج در این لینک آنلاین قرار دارد. ماتریس های درهمریختگی به ازای مرتبه ۱ الی ۱۰ به ترتیب به صورت زیر با روش TSNE حاصل شده است:



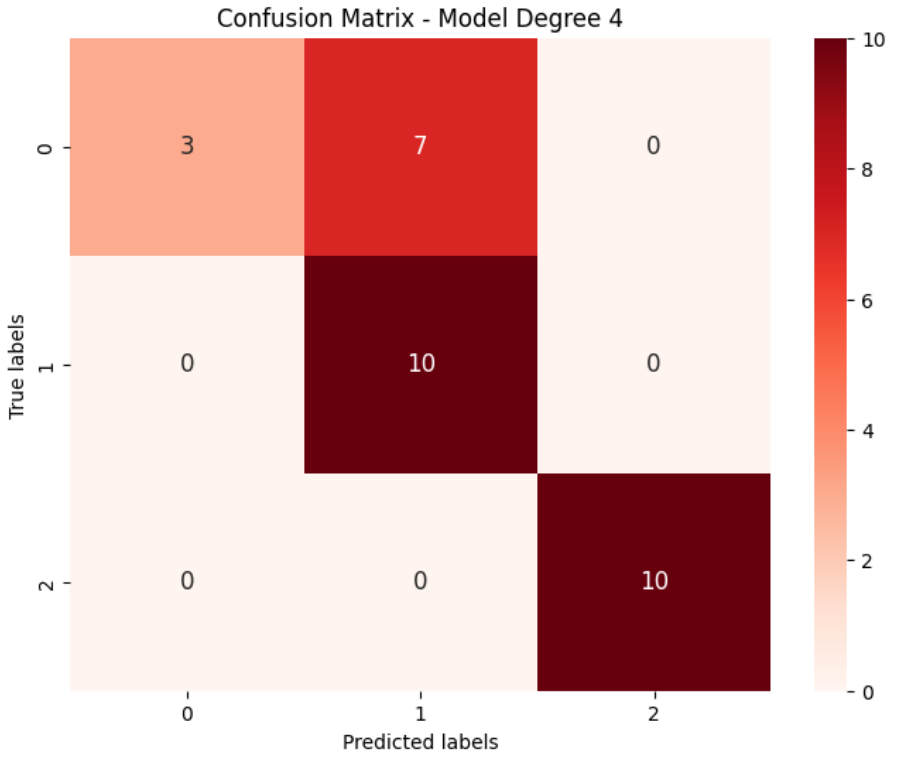
شکل ۷: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱



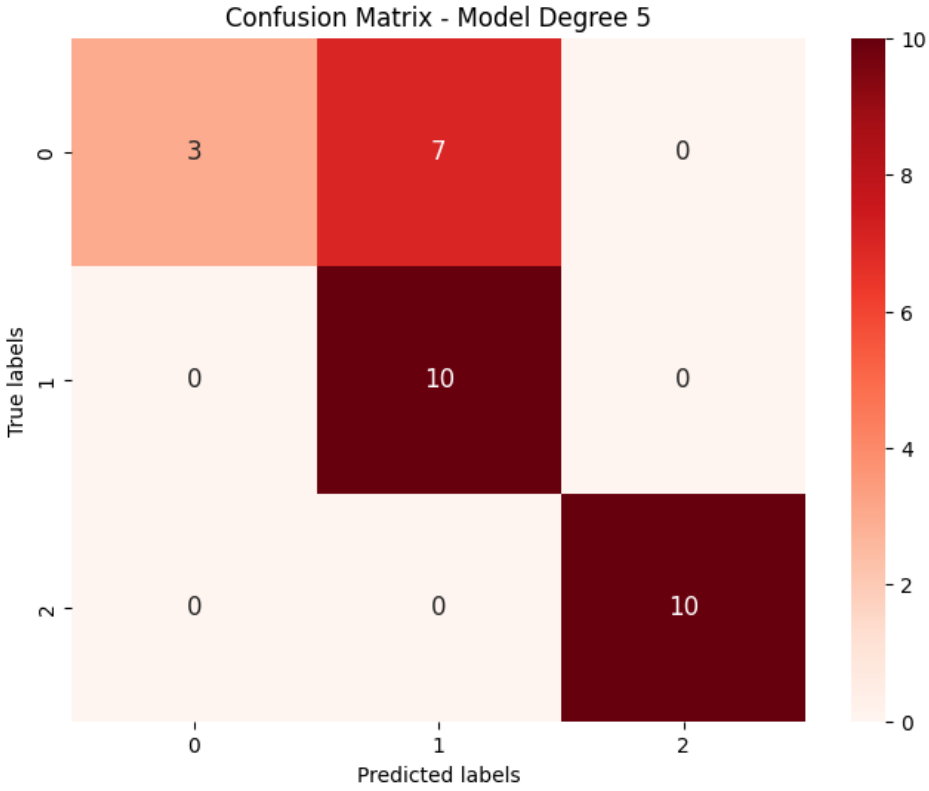
شکل ۸: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱



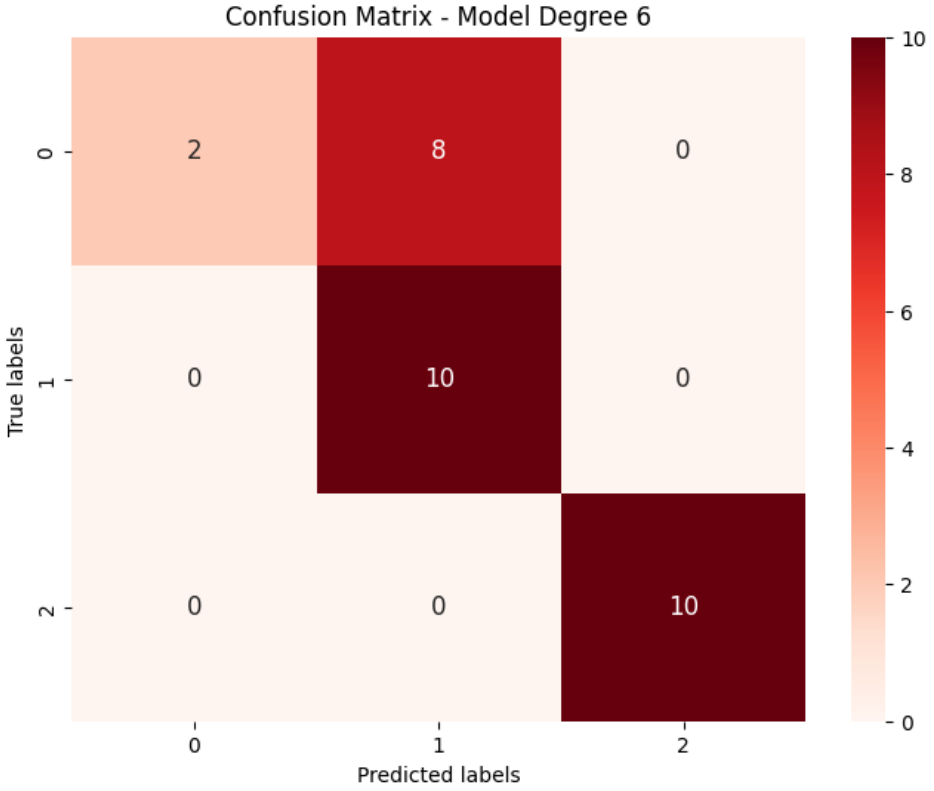
شکل ۹: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱



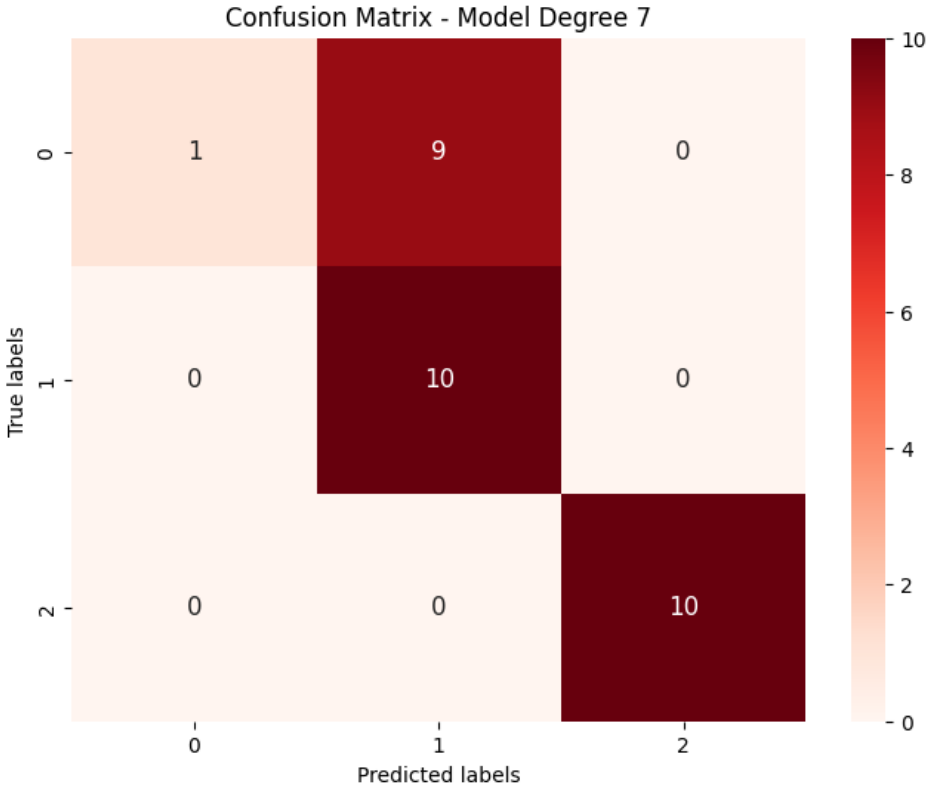
شکل ۱۰: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱



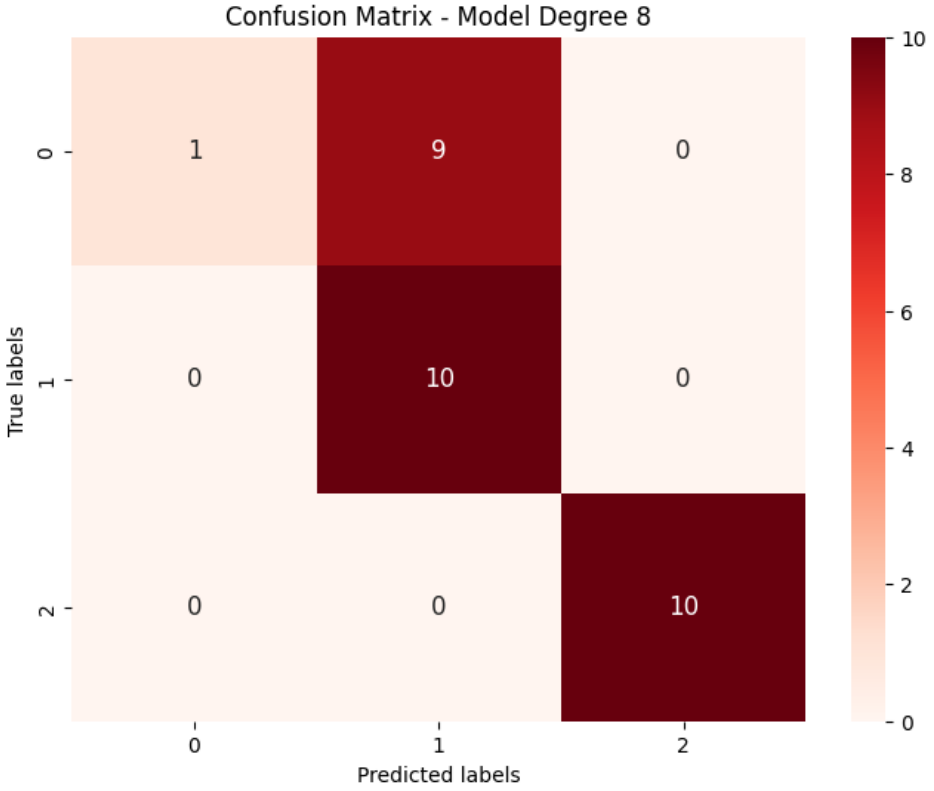
شکل ۱۱: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱



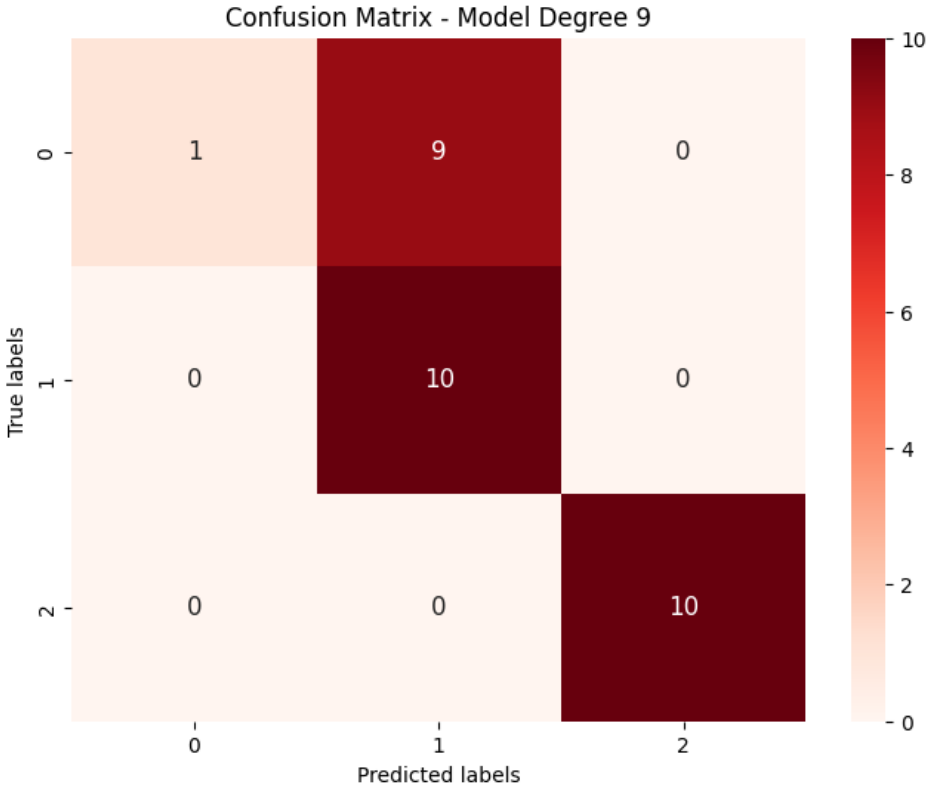
شکل ۱۲: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱



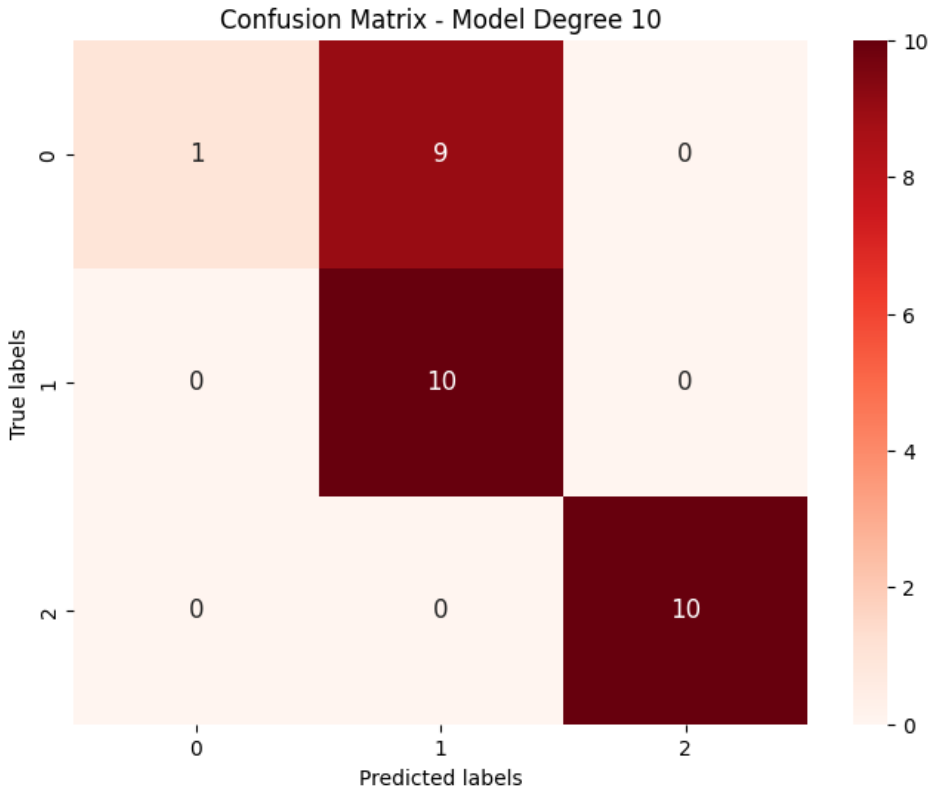
شکل ۱۳: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱



شکل ۱۴: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱

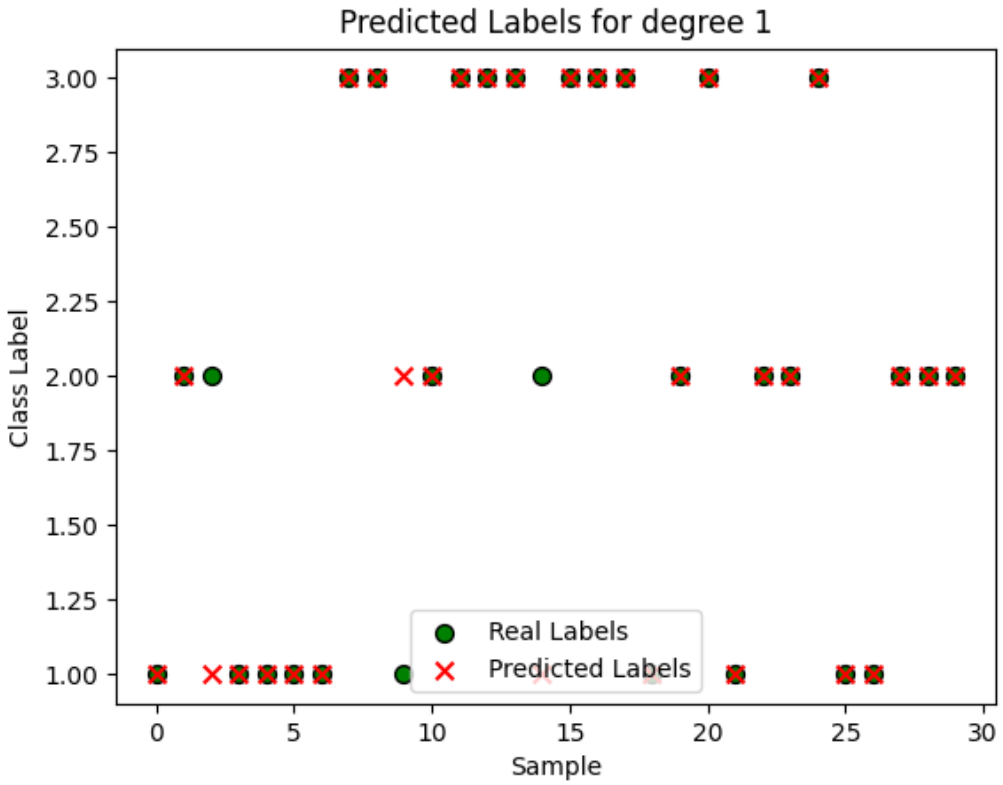


شکل ۱۵: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱



شکل ۱۶: ماتریس درهمریختگی به ازای کرنل چند جمله ای مرتبه ۱

طبق ماتریس های درهم ریختگی نشان داده شده در شکل های ۷ الی ۱۶ مشاهده می شود درابتدا که درجه کرنل چند جمله ای ۱ و ۲ است، دقت ۹۰ درصد به بالاتر است و فقط ۱ الی ۲ نمونه اشتباهی تفکیک شده است و به تدریج با افزایش درجه چند جمله ای مشاهده می شود بازدهی مدل خیلی پایین آمده به نحوی که به ازای درجه ۱۰ ، ۹ تا از داده های آزمون کلاس اول را اشتباهی در کلاس دوم تفکیک کرده است. موقعیت کلاس ها به ازای درجه های مختلف ۱ الی ۱۰ به صورت زیر است:



شکل ۱۷: gift ویژگی اصلی به ازای درجه ۱ الی ۱۰(لینک آنلاین)

جهت تبدیل عکس به Gift بالا ابتدا با دستور pip install imageio تولباکس imageio را نصب نموده سپس با قطعه کد زیر نتیجه خروجی هر ۱۰ مدل به ازای درجه های ذکر شده چاپ و ذخیره گردیده است:

```
import imageio

# List of model degrees
model_degrees = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Save the plot frames as a GIF
frames = []

# Iterate over model degrees
for degree in model_degrees:
    fig, ax = plt.subplots()
    # plt.clf()

    # Plot the real labels
    ax.scatter(range(len(y_test)), y_test, c='g', marker='o',
edgecolors='k', label='Real Labels', s=50)
    ax.set_xlabel('Sample Index')
    ax.set_ylabel('Class Label')
    ax.set_title('Real Labels')
```

```
    # Plot the predicted labels
    ax.scatter(range(len(y_pred2)), globals()[f"y_pred{degree}"],
c='r', marker='x', label='Predicted Labels', s=50)
    ax.set_xlabel('Sample')
    ax.set_ylabel('Class Label')
    ax.set_title(f'Predicted Labels for degree {degree}')
    ax.legend()
    # plt.clf()

    # Save the current figure to a buffer
    plt.draw()
    # plt.clf()

    # Convert the figure to a numpy array
    buf = np.frombuffer(fig.canvas.tostring_rgb(),
dtype=np.uint8)
    buf = buf.reshape(fig.canvas.get_width_height()[::-1] + (3,))

    # Append the buffer to the frames list
    frames.append(buf)

    # Clear the current figure

    plt.close(fig)
    plt.clf()


# Save the frames as a GIF
imageio.mimsave('predicted_labels.gif', frames, fps=.5)
plt.show()
```

د. حال الگوریتم SVM را برای مورد قبلی، بدون اســـتفاده از کتابخانهscikit-learn و به صـــورت From Scratchپیاده سازی کنید. در این بخش لازم است که یک کلاس SVM تعریف کنید. این کلاس می بایست حداقل دارای سـه تابع (متد) kernel_Polynomial ، Fitو Predict باشـد. متد kernel_Polynomial می بایســت با دریافت درجه های ۱ تا ۱۰، هســته های چندجمله ای را محاســبه کند. دقت الگوریتم را با افزایش درجه گزارش کنید و نتایج حاصـل را با بخش قبلی مقایسـه کنید. در این قسـمت نیز جداسـازی ویژگی های اصلی را برای درجات ۱ تا ۱۰ در قالب یک GIF به تصویر بکشـید پیوند دسـترسـی مسـتقیم آن را در گزارش خود قرار دهید.

حل)

در این قسـمت با اسـتفاده از یک کلاس و نوشـتن کد SVM به صـورت زیر و با کرنل چندجمله ای کار طبقه بندی گل زنبق را بررسی می کنیم. قطعه کد به فرم زیر به ازای درجه های ۱ الی ۱۰ نوشته شده است:

```
def linear_kernel( x1, x2):
```

```
    return np.dot(x1, x2)

def polynomial_kernel( x, y, C=1.0, d=3):
    return (np.dot(x, y) + C) ** d



def SVM1(X, X_t, y, C, kernel_type, poly_params=(1, 4)):
    kernel_and_params = (kernel_type, poly_params, C)
    n_samples, n_features = X.shape
    # Compute the Gram matrix
    K = np.zeros((n_samples, n_samples))
    if kernel_type == 'linear':
        for i in range(n_samples):
            for j in range(n_samples):
                K[i, j] = linear_kernel(X[i], X[j])
    elif kernel_type == 'polynomial':
        for i in range(n_samples):
            for j in range(n_samples):
                K[i, j] = polynomial_kernel(X[i], X[j],
poly_params[0], poly_params[1])
    else:
        raise ValueError("Invalid kernel type")

    # construct P, q, A, b, G, h matrices for CVXOPT
    P = cvxopt.matrix(np.outer(y, y) * K)
    q = cvxopt.matrix(np.ones(n_samples) * -1)
    A = cvxopt.matrix(y, (1, n_samples))
    b = cvxopt.matrix(0.0)
    G = cvxopt.matrix(np.vstack((np.diag(np.ones(n_samples) * -
1), np.identity(n_samples))))
    h = cvxopt.matrix(np.hstack((np.zeros(n_samples),
np.ones(n_samples) * C)))
    # solve QP problem
    cvxopt.solvers.options['show_progress'] = False
    solution = cvxopt.solvers.qp(P, q, G, h, A, b)
    # Lagrange multipliers
    a = np.ravel(solution['x'])
    # Support vectors have non-zero Lagrange multipliers
    sv = a > 1e-6  # some small threshold

    # Support vectors have non-zero Lagrange multipliers
    ind = np.arange(len(a))[sv]
    a = a[sv]
    sv_x = X[sv]
    sv_y = y[sv]
    numbers_of_sv = len(sv_y)
```

```
    # Bias (For linear it is the intercept):
    bias = 0
    if len(a) > 0:
        for n in range(len(a)):
            # For all support vectors:
            bias += sv_y[n]
            bias -= np.sum(a * sv_y * K[ind[n], sv])
        bias = bias / len(a)
    else:
        print("No support vectors found")
        bias = 0

    # Weight vector
    if kernel_type == 'linear':
        w = np.zeros(n_features)
        for n in range(len(a)):
            w += a[n] * sv_y[n] * sv_x[n]
    else:
        w = None

    y_pred = 0
    # Create the decision boundary for the plots. Calculates the
hypothesis.
    if w is not None:
        y_pred = np.sign(np.dot(X_t, w) + bias)
    else:
        y_predict = np.zeros(len(X_t))
        for i in range(len(X_t)):
            s = 0
            for a1, sv_y1, sv1 in zip(a, sv_y, sv_x):
                # a : Lagrange multipliers, sv : support vectors.
                # Hypothesis: sign(sum^S a * y * kernel + b)
                if kernel_type == 'linear':
                    s += a1 * sv_y1 * linear_kernel(X_t[i], sv1)
                if kernel_type == 'polynomial':
                    s += a1 * sv_y1 * polynomial_kernel(X_t[i],
sv1, poly_params[0], poly_params[1])
            y_predict[i] = s
        y_pred = np.sign(y_predict + bias)

    return w, bias, solution, a, sv_x, sv_y, y_pred,
kernel_and_params

def multiclass_svm(X,X_t, y, C, kernel_type, poly_params=(1, 4)):

    # Step 1: Identify unique class labels
```

```
    class_labels = list(set(y))

    # Step 2: Initialize classifiers dictionary
    classifiers = {}
    w_catch={} #catching w, b only for plot part
    b_catch={}
    a_catch={}
    sv_x_catch={}
    sv_y_catch={}
    # Step 3: Train binary SVM models for each required class
combination
    for i,class_label in enumerate(class_labels):
        # Create binary labels for current class vs. all others
        binary_y = np.where(y == class_label, 1.0, -1.0)
        # Train SVM classifier for binary classification
        w, bias, _,a, sv_x, sv_y,prediction,
kernel_and_params=SVM1(X,X_t, binary_y,
C,kernel_type,poly_params)
        classifiers[class_label] = prediction
        w_catch[class_label]=w
        b_catch[class_label]=bias
        a_catch[class_label]=a
        sv_x_catch[class_label]=sv_x
        sv_y_catch[class_label]=sv_y

    def decision_function(X_t):
        decision_scores = np.zeros((X_t.shape[0],
len(class_labels)))
        for i, label in enumerate(class_labels):
            decision_scores[:, i] = classifiers[label]
        return np.argmax(decision_scores,
axis=1),kernel_and_params,w_catch, b_catch,classifiers
    return decision_function(X_t)

def visualize_multiclass_classification(X_train, y_train1,
kernel_type, trainset, classifiers, class_labels, w_stack,
b_stack,kernel_and_params,degrees):
    plt.figure(figsize=(8, 6))
    (_,poly_params,C) = kernel_and_params
    # Plotting data points for each class
    for i, target_name in enumerate(class_labels):
        plt.scatter(X_train[y_train1 == i, 0], X_train[y_train1
== i, 1], label=target_name)

    if kernel_type == 'linear':

        h = .02  # step size in the mesh
```

```
        x_min, x_max = X_train[:, 0].min() - 1, X_train[:,
0].max() + 1
        y_min, y_max = X_train[:, 1].min() - 1, X_train[:,
1].max() + 1
        k=np.arange(x_min, x_max, h)
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
        x_test = np.c_[xx.ravel(), yy.ravel()]
        model=multiclass_svm(x_train,x_test, y_train,
C,kernel_type, poly_params)
        pred,_,_,_,_=model

        Z = pred.reshape(xx.shape)
        plt.contour(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

    else:
        h = .02  # step size in the mesh
        x_min, x_max = X_train[:, 0].min() - 1, X_train[:,
0].max() + 1
        y_min, y_max = X_train[:, 1].min() - 1, X_train[:,
1].max() + 1
        k=np.arange(x_min, x_max, h)
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
        x_test = np.c_[xx.ravel(), yy.ravel()]
        model=multiclass_svm(x_train,x_test, y_train,
C,kernel_type, poly_params)
        pred,_,_,_,_=model

        Z = pred.reshape(xx.shape)
        plt.contour(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)


    if trainset:
        plt.title(f'Data Points on Train Set {degrees}')
    else:
        plt.title(f'Data Points on Test Set {degrees}')

    plt.xlabel('PcA Component 1')
    plt.ylabel('PcA Component 2')
    plt.legend()
    plt.xlim(np.min(X_train[:, 0]) - 1, np.max(X_train[:, 0]) +
1)
    plt.ylim(np.min(X_train[:, 1]) - 1, np.max(X_train[:, 1]) +
1)
```

```
def calculate_metrics_and_plot(y_true, y_pred,degrees ,
labels=None):
    # Calculate confusion matrix
    cm = confusion_matrix(y_true, y_pred, labels=labels)

    # Calculate precision, recall, and F1-score
    precision = precision_score(y_true, y_pred, average=None,
labels=labels)
    recall = recall_score(y_true, y_pred, average=None,
labels=labels)
    f1 = f1_score(y_true, y_pred, average=None, labels=labels)

    # Calculate accuracy
    accuracy = np.sum(np.diag(cm)) / np.sum(cm)

    # Print precision, recall, F1-score, and accuracy
    for i in range(len(labels)):
        print(f"Class {labels[i]} - Precision:
{precision[i]:.4f}, Recall: {recall[i]:.4f}, F1-score:
{f1[i]:.4f}")

    print(f"Accuracy: {accuracy:.4f}")

    # Set custom color map and font size
    sns.set(font_scale=1.2)
    sns.set_style("whitegrid")

    # Plot the confusion matrix as a heatmap
    plt.figure(figsize=(8, 6))
    heatmap = sns.heatmap(cm, annot=True, fmt='d', cmap='BuGn',
xticklabels=labels, yticklabels=labels, annot_kws={"size": 14})

    # Set the title font
    heatmap.set_title(f'Confusion Matrix {degrees}',
fontdict={'fontsize': 16, 'family': 'serif'})

    plt.xlabel('Predicted')
    plt.ylabel('True')

    return accuracy
```

برنامه فوق بدنه اصلی کد با استفاده از حل به روش Q.P حل می گردد و با استفاده از قطعه کد زیر آموزش مدل به ازای درجه های ۱ الی ۱۰ و کرنل چند جمله ای صورت می گیرد:

```
degrees = range(1, 11)
```

```
acc = []
images_train = []
images_test = []
images_conf = []
# Train SVM with polynomial kernels of degree 1 to 10
for degree in degrees:
    #*****here is the callable function *****--------------------
----------------------------------------
    # Split the data into training and test sets
    x_train = Xn_train
    x_test = Xn_test
    y_test = yn_test
    y_train = yn_train

    model=multiclass_svm(x_train,x_test, y_train,
degree,'polynomial', poly_params=(1, degree))
    pred, kernel_and_params,w_catch, b_catch, classifiers=model

    #*****here is the callable function *****--------------------
----------------------------------------
    class_0 = 0
    class_1 = 1
    class_2 = 2

    plt.figure()
    visualize_multiclass_classification(x_train, y_train,
kernel_and_params[0], True, classifiers,
iris.target_names[class_0:class_2+2], w_catch, b_catch,
kernel_and_params, degree)
    filename1 = f'svm_degree_{degree}_train_support_vector.png'
    plt.savefig(filename1)
    images_train.append(imageio.imread(filename1))
    plt.close()

    plt.figure()
    visualize_multiclass_classification(x_test, y_test,
kernel_and_params[0], False, classifiers,
iris.target_names[class_0:class_2+2], w_catch, b_catch,
kernel_and_params, degree)
    filename2 = f'svm_degree_{degree}_test_support_vector.png'
    plt.savefig(filename2)
    images_test.append(imageio.imread(filename2))
    plt.close()

    print(iris.target_names[class_0:class_2+2])

    # Evaluate the model on the training set
```

```
    if __name__ == "__main__":
        plt.figure()
        y_true = y_test
        y_pred = pred
        print(f"Degree: {degree}\n")
        acc.append(calculate_metrics_and_plot(y_true, y_pred,
degree, labels=[0, 1, 2]))
        filename3 = f'svm_degree_{degree}_Confusion_Matrix.png'
        plt.savefig(filename3)
        images_conf.append(imageio.imread(filename3))
        plt.close()

# Path to save the GIF
gif_path_train =
'svm_poly_degrees_train_support_vector_fromsh.gif'
imageio.mimsave(gif_path_train, images_train,  fps=.5)
# Display the path to the GIF
print(f"GIF saved as {gif_path_train}")

# Path to save the GIF
gif_path_test = 'svm_poly_degrees_test_support_vector_fromsh.gif'
imageio.mimsave(gif_path_test, images_test, fps=.5)
# Display the path to the GIF
print(f"GIF saved as {gif_path_test}")

# Path to save the GIF
gif_path_conf = 'svm_poly_degrees_Confusion_Matrix_fromsh.gif'
imageio.mimsave(gif_path_conf, images_conf ,  fps=.5)
# Display the path to the GIF
print(f"GIF saved as {gif_path_conf}")
```

در این حالت انتظار می رود مطابق با بخش های قبل مدل به دلیل آنکه کاهش بعد با اسـتفاده از روش PCA انجام شده است به خوبی کلاس ها را تفکیک کند که به ازای درجه های مختلف کرنل چند جمله ای بازدهی به صورت زیر بدست آمده است:

به ازای مرتبه ۱ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 1

Class 0 - Precision: 0.6250, Recall: 1.0000, F1-score: 0.7692
Class 1 - Precision: 0.6250, Recall: 0.4167, F1-score: 0.5000
Class 2 - Precision: 0.8333, Recall: 0.6250, F1-score: 0.7143
Accuracy: 0.6667
```

به ازای مرتبه ۲ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 2

Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 0.9167, Recall: 0.9167, F1-score: 0.9167
```

```
Class 2 - Precision: 0.8750, Recall: 0.8750, F1-score: 0.8750
Accuracy: 0.9333
```

به ازای مرتبه ۳ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 3

Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 0.9167, Recall: 0.9167, F1-score: 0.9167
Class 2 - Precision: 0.8750, Recall: 0.8750, F1-score: 0.8750
Accuracy: 0.9333
```

به ازای مرتبه ۴ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 4

Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 0.9167, Recall: 0.9167, F1-score: 0.9167
Class 2 - Precision: 0.8750, Recall: 0.8750, F1-score: 0.8750
Accuracy: 0.9333
```

به ازای مرتبه ۵ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 5

Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 0.9091, Recall: 0.8333, F1-score: 0.8696
Class 2 - Precision: 0.7778, Recall: 0.8750, F1-score: 0.8235
Accuracy: 0.9000
```

به ازای مرتبه ۶ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 6

Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 1.0000, Recall: 0.8333, F1-score: 0.9091
Class 2 - Precision: 0.8000, Recall: 1.0000, F1-score: 0.8889
Accuracy: 0.9333
```

به ازای مرتبه ۷ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 7

Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 1.0000, Recall: 0.8333, F1-score: 0.9091
Class 2 - Precision: 0.8000, Recall: 1.0000, F1-score: 0.8889
Accuracy: 0.9333
```

به ازای مرتبه ۸ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 8

Class 0 - Precision: 0.9091, Recall: 1.0000, F1-score: 0.9524
Class 1 - Precision: 1.0000, Recall: 0.8333, F1-score: 0.9091
Class 2 - Precision: 0.8889, Recall: 1.0000, F1-score: 0.9412
Accuracy: 0.9333
```

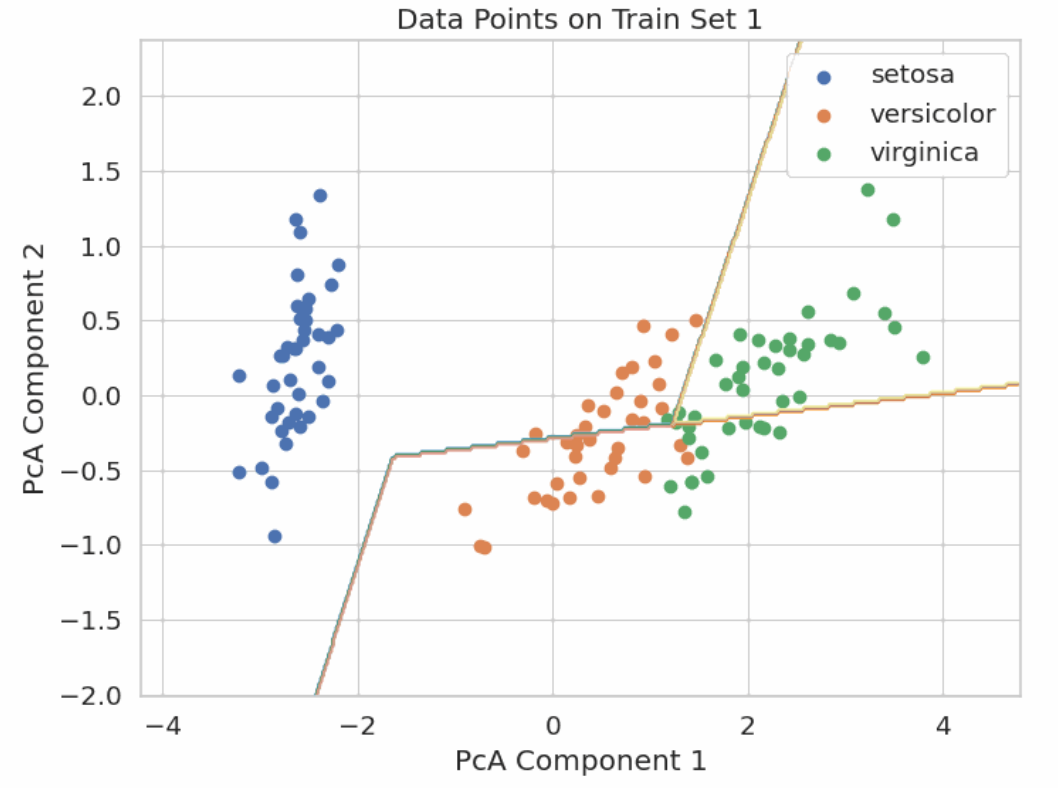
به ازای مرتبه ۹ نتایج بازدهی به این صورت حاصل شده است:

```
Degree: 9

Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 1.0000, Recall: 0.9167, F1-score: 0.9565
Class 2 - Precision: 0.8889, Recall: 1.0000, F1-score: 0.9412
Accuracy: 0.9667
```

به ازای مرتبه ۱۰ نتایج بازدهی به این صورت حاصل شده است:

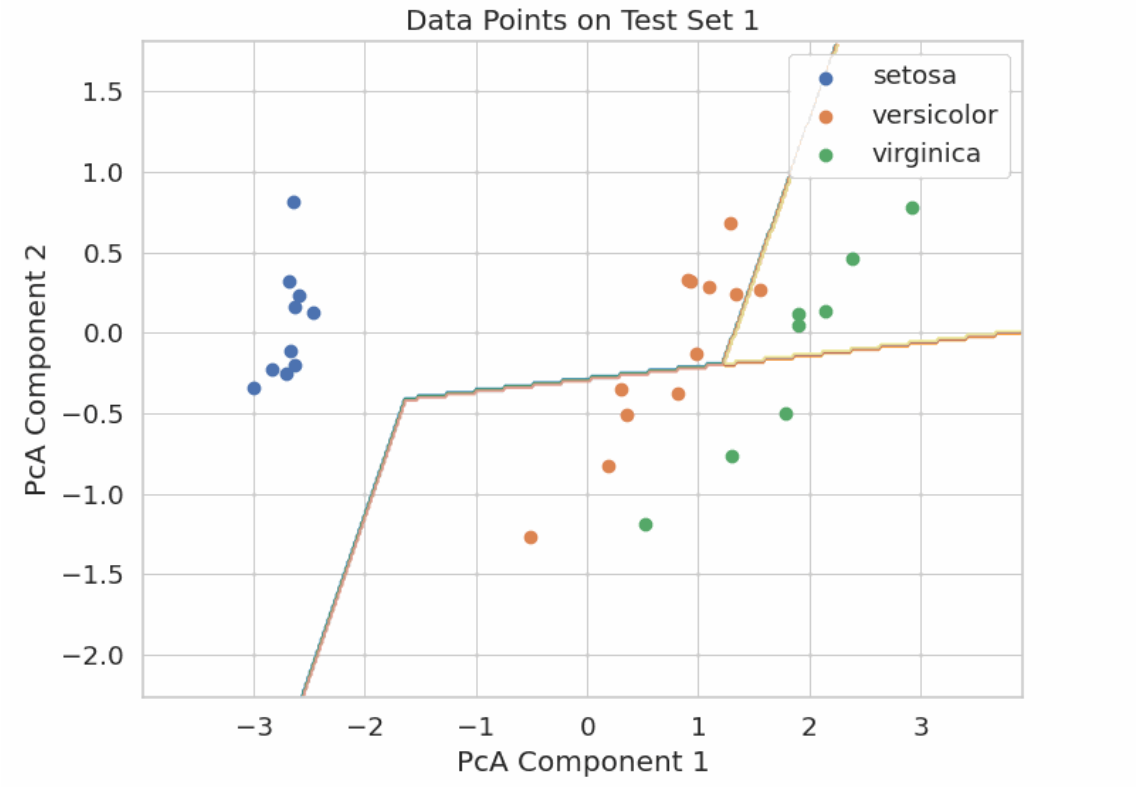
```
Degree: 10

Class 0 - Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000
Class 1 - Precision: 1.0000, Recall: 0.8333, F1-score: 0.9091
Class 2 - Precision: 0.4000, Recall: 1.0000, F1-score: 0.5714
Accuracy: 0.6000
```

ناحیه تصمیم برای داده های آزمایش به صورت شکل زیر حاصل گردیده است:



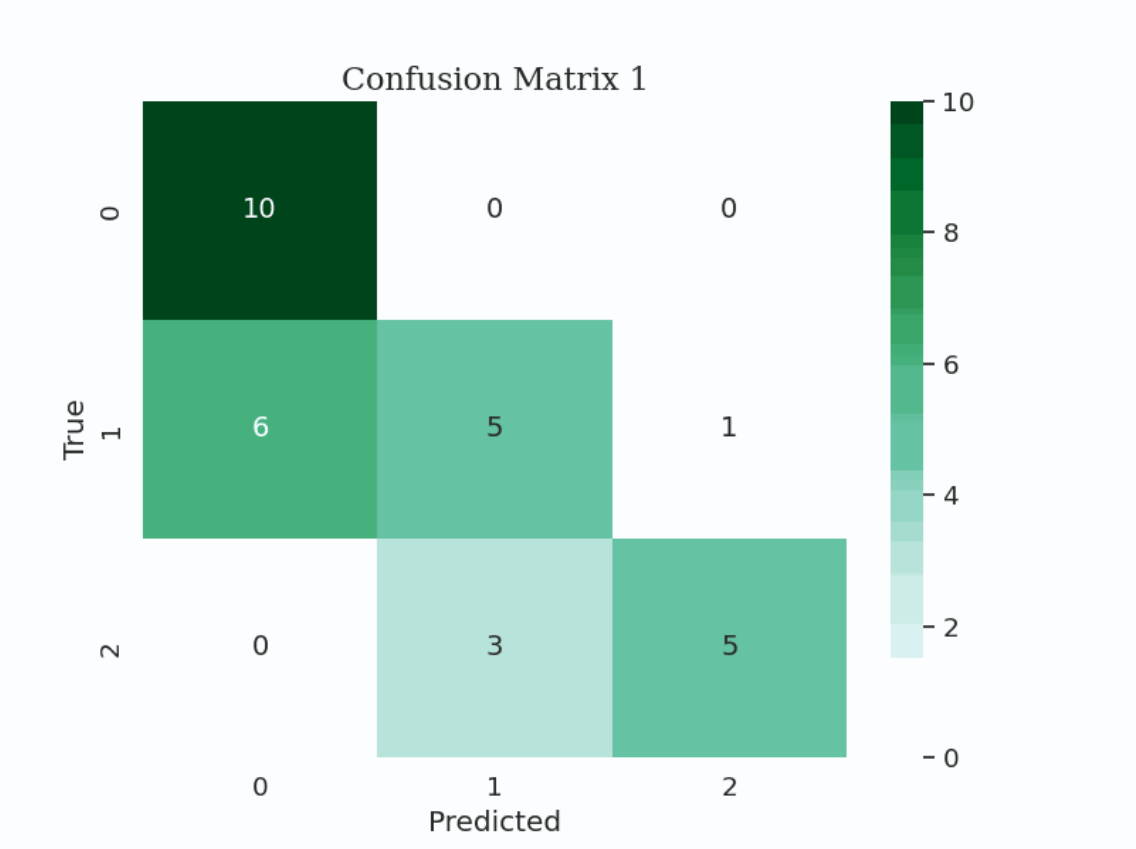
شکل ۱۸: ناحیه تصمیم داده های آزمایش با استفاده از Scratch SVM (لینک آنلاین)

ناحیه تصمیم برای داده های آزمون نیز به ازای درجه های ۱ الی ۱۰ به صورت شکل زیر حاصل گردیده است:



شکل ۱۹: ناحیه تصمیم داده های ارزیابی با استفاده از Scratch SVM (لینک آنلاین)

همچنین ماتریس درهم ریختگی به ازای درجه های ۱ الی ۱۰ نیز به صورت شکل زیر بدست آمده است:



شکل ۲۰: ماتریس درهم ریختگی با استفاده از Scratch SVM (لینک آنلاین)

طبق نتایج بدست آمده در این بخش و بخش های قبلی مشاهده می گردد به ازای درجه های چند جمله ای مرتبه پایین(عموما ۱ الی ۳) مدل عملکرد خوبی دارد و با دقت بالای ۹۰ درصدی کار تفکیک کلاس های داده را به خوبی انجام می دهد. اما با افزایش مرتبه درجه چند جمله ای و پیچیده شدن آن مشاهد می شود دقت مدل به دلیل آموزش بیش از حد بسیار پایین آمده است و به ازای درجه ۱۰ به عنوان مثال میزان دقت به ۶۰ کاهش پیدا کرده است. بنابراین با مقایسه این نتایج می توان نتیجه گرفت که تا حد امکان نباید مدل را پیچیده نموده و با ساده ترین مدل ماشین را باید آموزش دهیم.

شایان ذکر است کد پایتون این قسمت در این لینک است.

فصل ۲

سوال دوم

۲-۱- مقاله Credit Card Fraud Detection Using Autoencoder Neural Network برای پیاده سازی در این قسمت در نظر گرفته شده است. پس از مطالعهٔ مقاله به سوالات زیر پاسخ دهید.

آ. بزرگ ترین چالش ها در توسعهٔ مدل های تشخیص تقلب چیست؟ این مقاله برای حل این چالش ها از چه روش هایی استفاده کرده است؟

پاسخ:

طبق مقاله، بزرگترین چالش در توسعه مدل‌های تشخیص تقلب، **عدم تعادل داده‌ها** است. در این حالت، تعداد نمونه‌های کلاس اقلیت (معاملات تقلبی) بسیار کمتر از نمونه‌های کلاس اکثریت (معاملات واقعی) است. این عدم تعادل می‌تواند منجر به مدل‌هایی شود که به طور تصادفی بر روی کلاس اکثریت تمرکز می‌کنند و معاملات تقلبی را به درستی شناسایی نمی‌کنند. در این مقاله، برای حل این چالش از روش **شبکه عصبی خودکار رمزگذار نویزی** *(Denoising Autoencoder Neural Network)* استفاده شده است. این شبکه عصبی می‌تواند ویژگی‌های کلیدی داده‌ها را یاد گرفته و معاملات تقلبی را از معاملات واقعی تشخیص دهد.

ب. در مورد معماری شبکهٔ ارائه شده در مقاله به صورت مختصر توضیح دهید.

پاسخ:

طبق مقاله، مدل شبکه عصبی مورد استفاده یک **شبکه عصبی خودکار رمزگذار نویزدار** *(Denoising Autoencoder Neural Network)* است. این مدل نوعی از شبکه عصبی خودکار رمزگذار است که برای یادگیری از داده‌های ناقص یا نویزی در آن استفاده می‌شود.

شبکه عصبی خودکار رمزگذار نویزدار شامل دو بخش اصلی است:

- **رمزگذار** ***(Encoder):*** این بخش وظیفه فشرده‌سازی داده‌ها را بر عهده دارد. به عبارت دیگر، داده‌های ورودی را به یک نمایش فشرده *(representation)* با بعد کم تر از ورودی تبدیل می‌کند.
- **رمزگشا** ***(Decoder):*** این بخش وظیفه بازسازی داده‌ها از نمایش فشرده را بر عهده دارد. به عبارت دیگر، نمایش فشرده را به داده‌های اصلی تبدیل می‌کند.

در این مدل، داده‌های ورودی به شبکه عصبی رمزگذار وارد می‌شوند. رمزگذار داده‌ها را فشرده کرده و یک نمایش فشرده از آنها را تولید می‌کند. سپس، این نمایش فشرده به شبکه عصبی رمزگشا وارد می‌شود. رمزگشا تلاش می‌کند تا از این نمایش فشرده، داده‌های اصلی را بازسازی کند. در فرآیند آموزش، شبکه عصبی یاد می‌گیرد که چگونه داده‌های ورودی را به طور دقیق فشرده و بازسازی کند. این کار باعث می‌شود که شبکه عصبی ویژگی‌های کلیدی داده‌ها را یاد گرفته و بتواند آنها را از داده‌های ناقص یا نویزی تشخیص دهد. ساختار این شبکه ها در شکل های زیر آورده شده است.

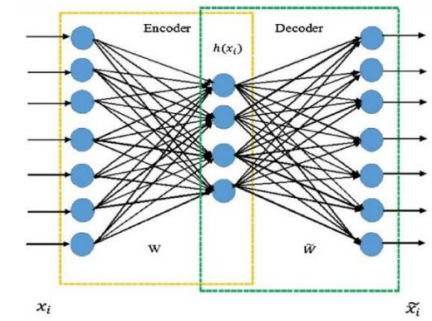


Fig. 1 architecture of autoencoder neural network

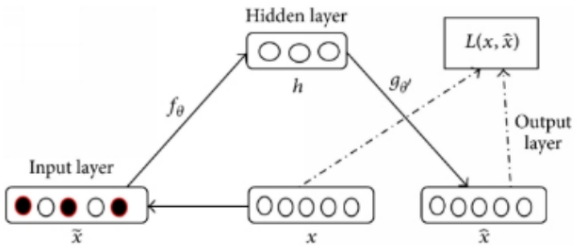


Fig. 2 Denoising autoencoder neural network

شکل ۲۱: ساختار Auto encoder NN

ج. مدل ارائه شده را پیاده سازی کرده و با استفاده از این دیتاست آموزش دهید. برای جلوگیری از بیش برازش، آموزش مدل را طوری تنظیم کنید که در انتهای آموزش، بهترین وزن های مدل بر اساس خطای قسمت اعتبارسنجی بازگردانده شود.

پاسخ:

دیتاست‌های تقلب با کارت اعتباری

دیتاست‌های تقلب با کارت اعتباری مجموعه‌ای از تراکنش‌های کارت اعتباری را شامل می‌شوند که به عنوان "مشروع" یا "تقلبی" برچسب‌گذاری شده‌اند .این دیتاست‌ها برای آموزش مدل‌های یادگیری ماشین جهت شناسایی تراکنش‌های تقلبی در دنیای واقعی استفاده می‌شوند.

- **ویژگی‌ها**

دیتاست‌های تقلب با کارت اعتباری شامل تعدادی از ویژگی‌ها هستند که اطلاعات مربوط به تراکنش را نشان می‌دهند .برخی از ویژگی‌های رایج عبارتند از:

- **مبلغ تراکنش :** مبلغی که در تراکنش خرج شده است.
- **تاریخ و زمان تراکنش :**تاریخ و زمان انجام تراکنش.
- **محل تراکنش :**مکانی که تراکنش در آن انجام شده است (مانند فروشگاه یا وب‌سایت).
- **نوع تراکنش :**نوع تراکنش انجام شده (مانند خرید، برداشت وجه نقد، یا انتقال وجه).

- **اطلاعات کارت :**اطلاعات مربوط به کارت اعتباری که برای انجام تراکنش استفاده شده است) مانند شماره کارت، تاریخ انقضا، و.(CVV
- **اطلاعات حساب :**اطلاعات مربوط به حساب بانکی مرتبط با کارت اعتباری (مانند نام صاحب حساب، آدرس، و شماره تلفن).
- **اطلاعات مربوط به دستگاه :**اطلاعات مربوط به دستگاهی که برای انجام تراکنش استفاده شده است)مانند نوع دستگاه، سیستم عامل، و آدرس.(IP
- **خروجی (برچسب‌ها)**

خروجی دیتاست‌های تقلب با کارت اعتباری یک برچسب است که نشان می‌دهد آیا تراکنش "مشروع" یا "تقلبی" است .این برچسب‌ها بر اساس قضاوت‌های انسانی یا قوانین از پیش تعیین‌شده تعیین می‌شوند.

دیتاست مربوطه در این بخش دارای بعد ۲۸ و ۲۸۴۸۰۷ نمونه است که با دستور زیر داده ها فراخوانی شده است:

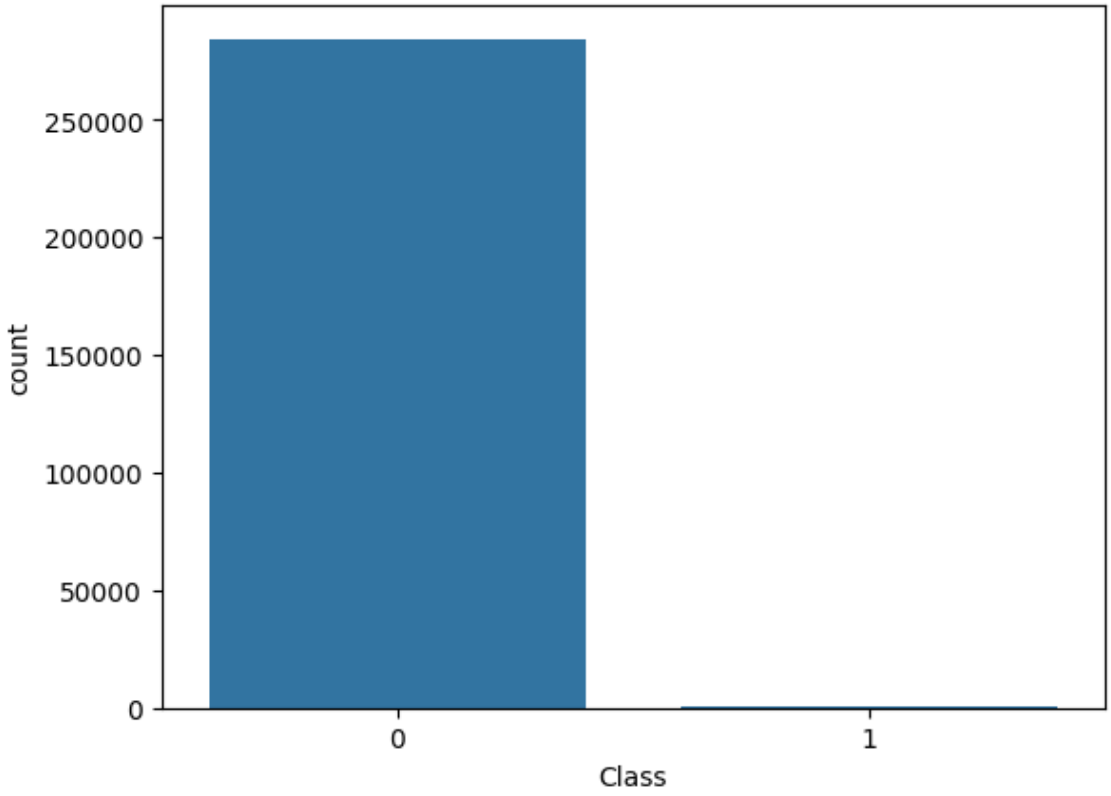
```
data = pd.read_csv('creditcard.csv')
X = data.drop('Class', axis=1).values
y = data['Class'].values
```

با استفاده از پیش پردازش اولیه و دستور ()null وجود دیتا null بررسی شده و طبق گزارش زیر نبوده است:

```
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
```

```
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
```

با استفاده از دستور sns.countplot(x='Class',data=data) توازن کلاس های داده ها به صورت شکل زیر بدست آمده است:

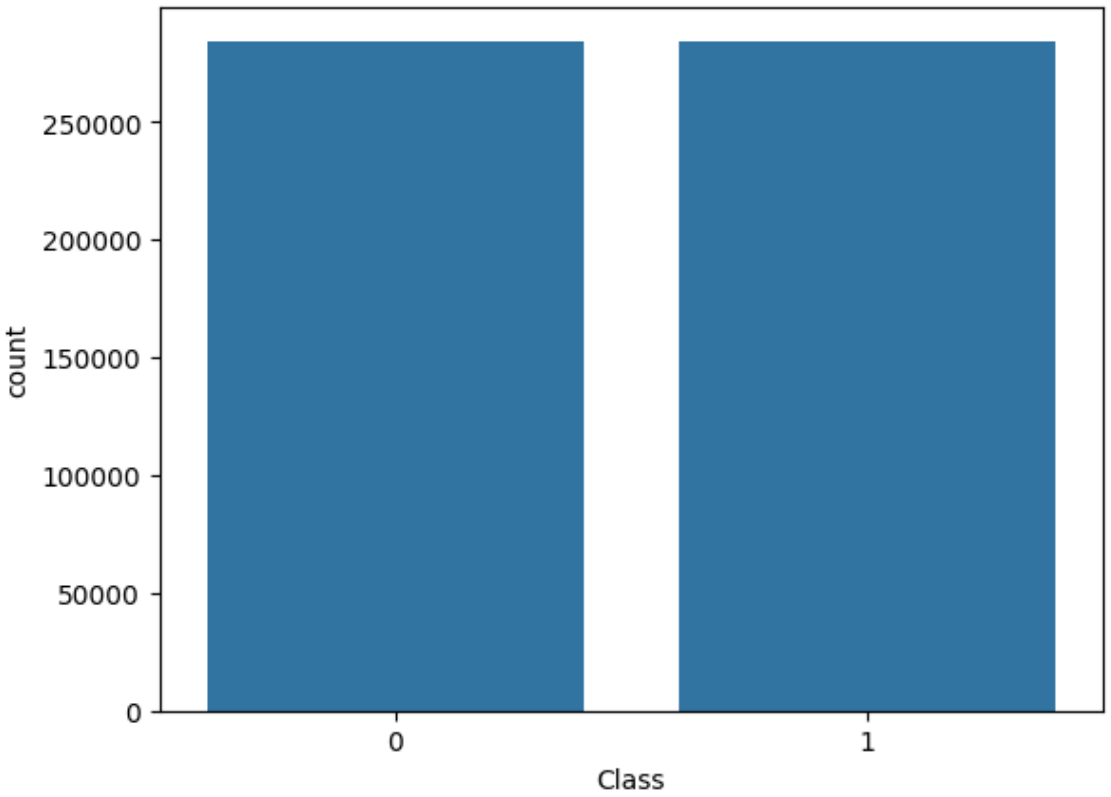


شکل ۲۲: شکل میزان توازن در کلاس داده های تقلبی و مشروح

مشاهده می شود طبق شکل که در داده های کلاس تقلب و مشروح میزان توازن برقرار نیست و تعداد داده های کلاس مشروح بسیار بیشتر است و جهت اینکه ماشین هر دو کلاس را بخوبی آموزش ببیند و کلاس صفر یا مشروح را فقط یاد نگیرد داده ها را با دستور زیر متوازن می کنیم:

```
df_resampled = pd.DataFrame(X_resampled, columns=data.columns[:-
1])
df_resampled['Class'] = y_resampled

sns.countplot(x='Class',data=df_resampled)
```

نتیجه اجرای این دستور که داده های متوازن در هر دو کلاس است به صورت شکل زیر است:



شکل ۲۳: داده های هر دو کلاس متوازن شده

همچنین جهت آغشته کردن داده ها به نویز از قطعه کد زیر نویز با واریانس ۰.۲ را تولید می نماییم:

```
noise_factor = 0.2
X_train_noisy = X_train_scaled + noise_factor *
np.random.normal(loc=0.0, scale=1.0, size=X_train_scaled.shape)
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
X_test_noisy = X_test_scaled + noise_factor *
np.random.normal(loc=0.0, scale=1.0, size=X_test_scaled.shape)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)
```

در ادامه داده های آغشته به نویز را به شبکه Autoencoder با شش لایه که تابع loss آن بر اساس مجموع مربعات داده خام و خروجی Autoencoder است و کار حذف نویز در این بخش اتفاق می افتد که کد مربوطه و فرا پارامترهای مدل مانند توابع فعالساز، تعداد گام آموزشی و تعداد batch داده ها در آن مشخص شده است:

```
input_dim = X_train_noisy.shape[1]
encoding_dim = 10

input_layer = Input(shape=(input_dim,))
encoder = Dense(22, activation="relu")(input_layer)
encoder = Dense(15, activation="relu")(encoder)
encoder = Dense(encoding_dim, activation="relu")(encoder)
```

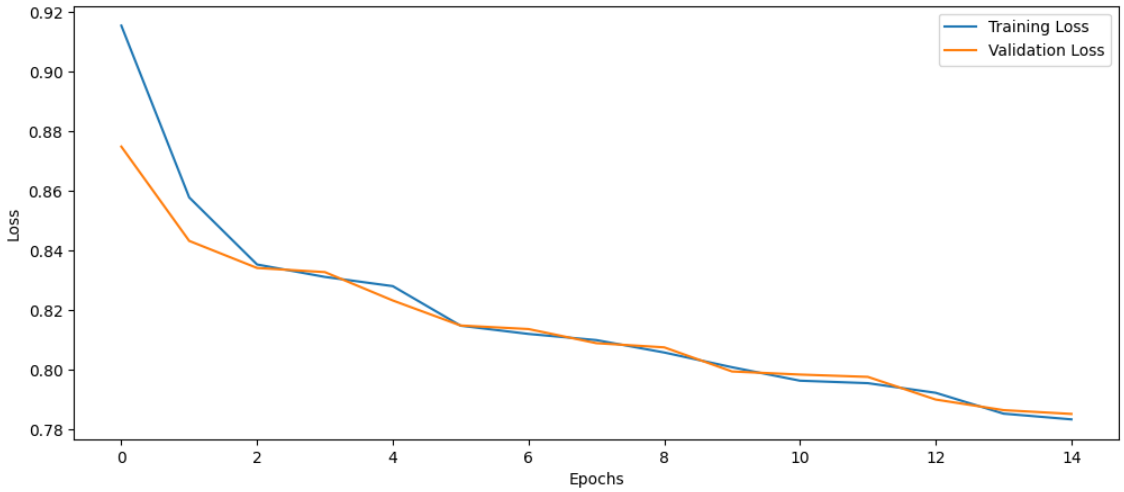
```
decoder = Dense(15, activation="relu")(encoder)
decoder = Dense(22, activation="relu")(decoder)
decoder = Dense(input_dim, activation="sigmoid")(decoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

model_checkpoint = ModelCheckpoint('best_autoencoder.h5',
monitor='val_loss', save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
verbose=1)
history_ae = autoencoder.fit(X_train_noisy, X_train_scaled,
                             epochs=15,
                             batch_size=512,
                             shuffle=True,
                             validation_split=0.2,
                             callbacks=[model_checkpoint,early_st
opping])

autoencoder.load_weights('best_autoencoder.h5')

X_train_denoised = autoencoder.predict(X_train_noisy)
X_test_denoised = autoencoder.predict(X_test_noisy)

plt.figure(figsize=(12, 5))
plt.plot(history_ae.history['loss'], label='Train Loss')
plt.plot(history_ae.history['val_loss'], label='Validation Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

print(" ----------------------------------------------classifier---
---------------------------------------- ")


input_layer = Input(shape=(input_dim,))
classifier = Dense(22, activation="relu")(input_layer)
classifier = Dense(15, activation="relu")(classifier)
classifier = Dense(10, activation="relu")(classifier)
classifier = Dense(5, activation="relu")(classifier)
classifier = Dense(2, activation="softmax")(classifier)

classifier_model = Model(inputs=input_layer, outputs=classifier)
classifier_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model_checkpoint_classifier =
ModelCheckpoint('best_classifier.h5', monitor='val_loss',
save_best_only=True, verbose=1)

history_clf = classifier_model.fit(X_train_denoised, y_train,
                                   epochs=15,
                                   batch_size=512,
                                   shuffle=True,
                                   validation_split=0.2,
                                   callbacks=[model_checkpoint_cl
assifier])

classifier_model.load_weights('best_classifier.h5')

y_pred = classifier_model.predict(X_test_denoised)
y_pred_classes = np.argmax(y_pred, axis=1)

rscore_1 = r2_score(y_test, y_pred_classes)
print("R2 Score: ", rscore_1)

plt.figure(figsize=(12, 5))
plt.plot(history_clf.history['loss'], label='Train Loss')
plt.plot(history_clf.history['val_loss'], label='Validation
Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

cf_matrix = confusion_matrix(y_test, y_pred_classes)
cf_matrix_percent = cf_matrix.astype('float') /
cf_matrix.sum(axis=1)[:, np.newaxis] * 100

plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_percent, annot=True, fmt='.2f',
cmap='Blues', annot_kws={"size": 12})

class_report = classification_report(y_test, y_pred_classes)
print("\nClassification Report:\n", class_report)
```

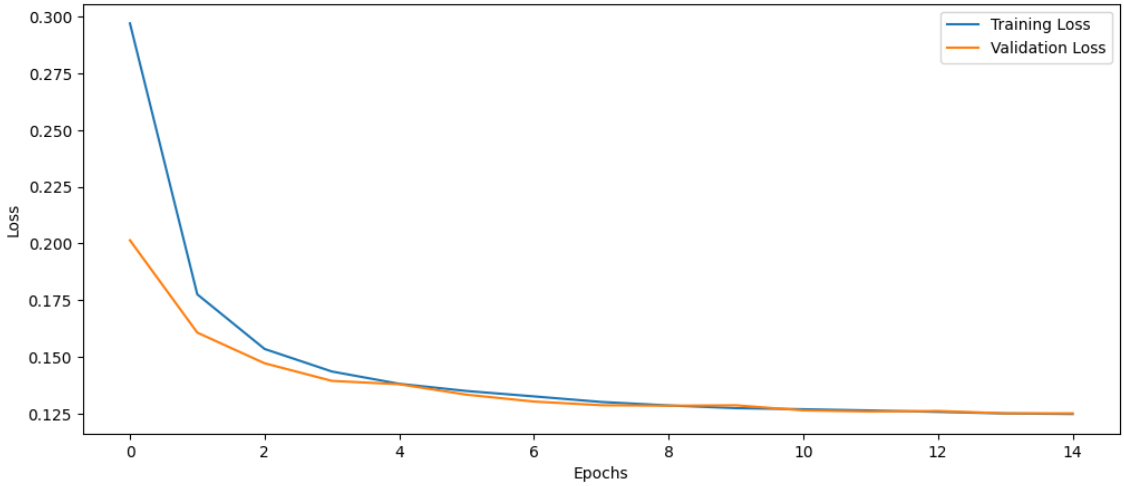
د. ماتریس درهم ریختگی را روی قسمت آزمون داده ها رسم کنید و مقادیر Accuracy ، Precision، Recall و f1 score را گزارش کنید. فکر می کنید در مسائلی که توزیع برچسب ها نامتوازن است، استفاده

از معیاری مانند Accuracyبه تنهایی عمل کرد مدل را به درستی نمایش می دهد؟ چرا؟ اگر نه، کدام معیار می تواند به عنوان مکمل استفاده شود؟

با run کردن کد آورده شده در بخش ج، مقدار شاخص های loss function مربوط به قسمت آموزش و ارزیابی بخش های Autoencoder و Classifier به ترتیب به صورت شکل های زیر حاصل شده اند:



شکل ۲۴: شاخص loss قسمت آموزش و ارزیابی Autoencoder



شکل ۲۵: شاخص loss قسمت آموزش و ارزیابی Classifier

طبق این دو شکل مشاهد می شود هم نمودار آموزش و هم نمودار آزمون تابع Loss اتوانکدر و طبقه بند روندی نزولی در پیش دارند. گزارش طبقه بندی مدل و نتایج ارزیابی توسط دستور

```
class_report = classification_report(y_test, y_pred_classes)
print("\nClassification Report:\n", class_report)
```

```
Classification Report:
              precision    recall  f1-score   support
```

0	0.94	0.98	0.96	56863
1	0.98	0.93	0.95	56863
accuracy			0.96	113726
macro avg	0.96	0.96	0.96	113726
weighted avg	0.96	0.96	0.96	113726

طبق این گزارش نتایج زیر قابل تحلیل است:

معیارهای ارزیابی

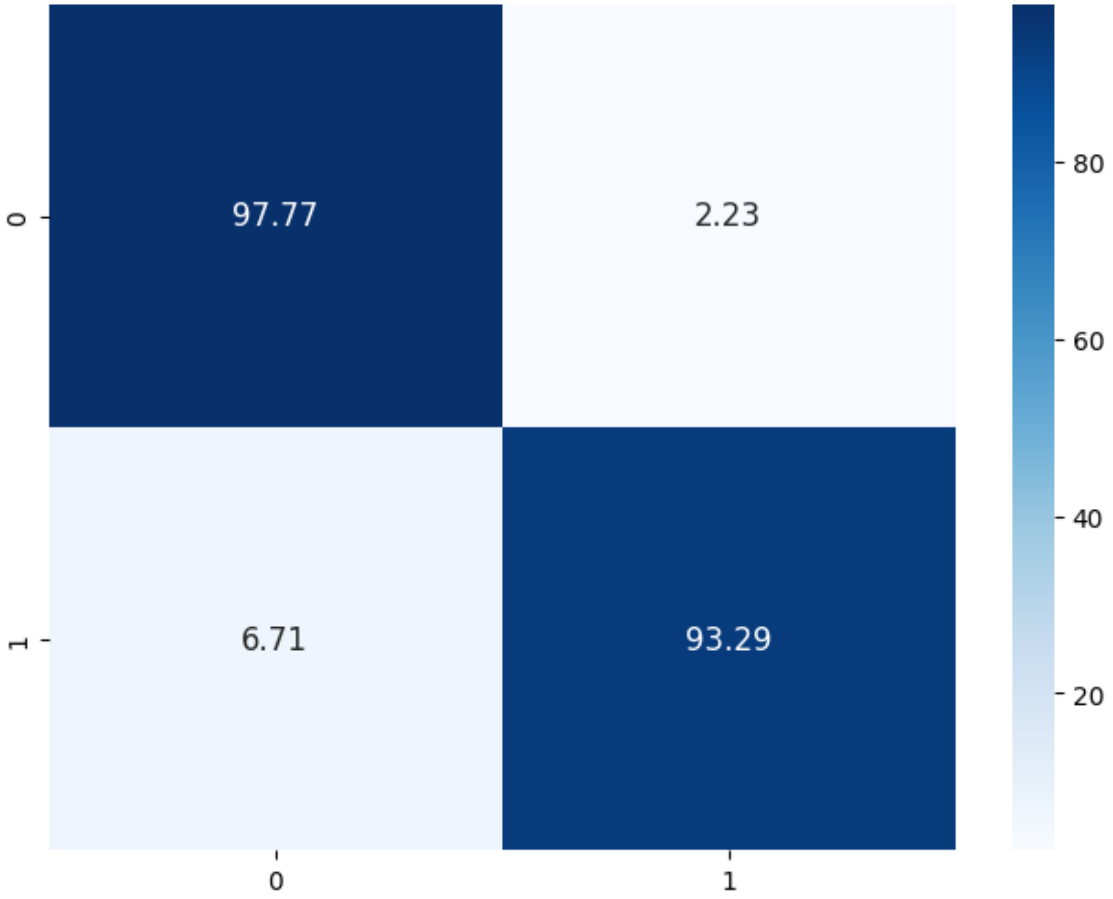
- **دقت :(Accuracy)** نسبت پیش‌بینی‌های صحیح به کل پیش‌بینی‌ها.
- **دقت :(Precision)** نسبت پیش‌بینی‌های صحیح کلاس ۱ به کل پیش‌بینی‌هایی که به عنوان ۱ پیش‌بینی شده‌اند.
- **بازخوانی :(Recall)** نسبت پیش‌بینی‌های صحیح کلاس ۱ به کل نمونه‌های واقعی کلاس ۱.
- **امتیاز :F1** میانگین هارمونیک دقت و بازخوانی.
- **تفسیر نتایج**

در این گزارش، دقت مدل ۹۶٪ است یعنی ۹۶ درصد داده های هر دو کلاس را که تعداد ۵۶۸۶۳ است را به خوبی تفکیک کرده است.

همچنین دقت و بازخوانی برای هر دو کلاس (۰ و ۱) بالا است .این نشان می‌دهد که مدل در تشخیص هر دو کلاس عملکرد خوبی دارد.

امتیاز F1 نیز برای هر دو کلاس بالا است که نشان می‌دهد مدل در تشخیص هر دو کلاس تعادل خوبی بین دقت و بازخوانی برقرار می‌کند.

ماتریس درهمریختگی نیز به صورت شکل زیر برای دو کلاس بالانس شده بدست آمده است:



شکل ۲۶: ماتریس درهمریختگی داده های کلاس تقلب و مشروح

طبق شکل ماتریس درهمریختگی مشاهده می شود مدل طبقه بند توانسته است با دقت تقریبا ۹۸ درصدی داده های کلاس مشروح و با دقتی حدود ۹۴ داده های کلاس تقلب را به خوبی از هم تفکیک کند.

دقت و عدم تعادل در توزیع برچسب‌ها

این درست است که Accuracy معیار ساده و قابل فهمی برای سنجش عملکرد مدل‌های طبقه‌بندی هست، اما در مواقعی که توزیع برچسب‌ها نامتوازن باشد، استفاده از این معیار به تنهایی می‌تونه گمراه‌کننده باشه.

- **چرا Accuracy گمراه‌کننده است؟**

فرض کنید دو کلاس داریم:**کلاس اکثریت** :شامل ۹۹٪ از نمونه‌ها، **کلاس اقلیت** :شامل ۱٪ از نمونه‌ها، اگر مدل همیشه کلاس اکثریت را پیش‌بینی کند، به Accuracy 99% می‌رسد. اما این به معنی عملکرد خوب مدل در تشخیص کلاس اقلیت نیست. در واقع، مدل هیچ تلاشی برای تشخیص کلاس اقلیت انجام نمی‌دهد.

- **معیارهای جایگزین**

برای ارزیابی عملکرد مدل در مسائلی که توزیع برچسب‌ها نامتوازن هست، معیارهای دیگری وجود دارند که Accuracyرو تکمیل می‌کنند:

Precision: نسبت پیش‌بینی‌های صحیح کلاس ۱ به کل پیش‌بینی‌هایی که به عنوان ۱ پیش‌بینی شده‌اند.

Recall: نسبت پیش‌بینی‌های صحیح کلاس ۱ به کل نمونه‌های واقعی کلاس ۱.

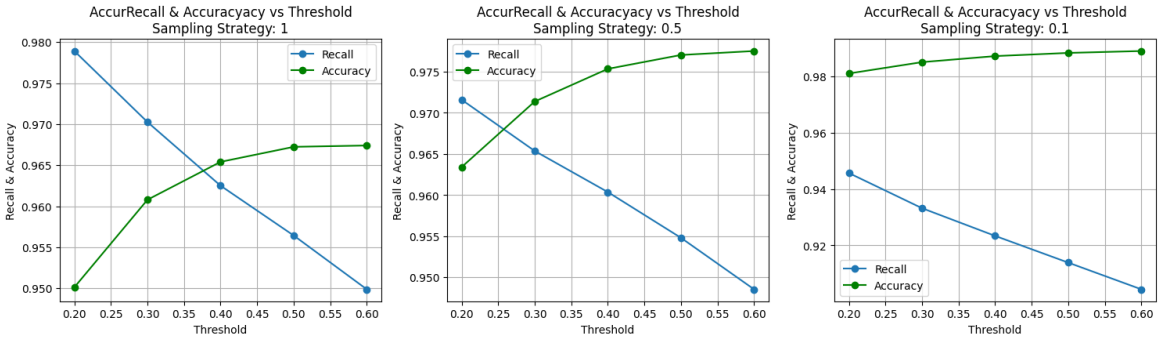
F1-score: میانگین هارمونیک دقت و بازخوانی.

AUC: (منحنی مشخصه عملگر) :نشان‌دهنده توانایی مدل در تشخیص نمونه‌های مثبت و منفی در سطوح مختلف آستانه.

استفاده از این معیارها به همراه Accuracy دید دقیق‌تری از عملکرد مدل در تشخیص هر دو کلاس ارائه می‌دهد.

ه. با آستانه های مختلف برای Oversampling عمل کرد مدل را بررسی کرده و نمودار Accuracy & Recallرا مانند شکل ۷ مقاله ترسیم کنید.

در این قسمت به ازای Oversampling های ۰.۱، ۰.۵ و ۱ مقدار نمودار Accuracy & Recall به صورت شکل زیر بدست آمده است:



شکل ۲۷: نمودار Accuracy و Recall

طبق شکل ملاحظه می شود به ازای Oversampling های مختلف نمودار Accuracy و Recall به دلیل آنکه مقدار داده ها متوازن است بسیار خوب و نزدیک به ۹۸ درصد است و این به آن دلیل است که classifier داده ها را به خاطر بالانسی خوب فرا گرفته است و در حالتی که اگر داده ها متوازن یا بالانس نباشند (حالتی که مقدار Oversampling به صفر نزدیکتر است) طبق نمودار سوم مشاهده می شود که نمودار recall بسیار کمتر خواهد شد.

و. مدل را با استفاده از داده های نامتوازن و بدون حذف نویز، آموزش داده و موارد بخش قبلی را گزارش کنید و نتایج دو مدل را با هم مقایسه کنید.

در این قسمت داده های نویزی و نامتوازن را بدون در نظر Autoencoder به طبقه بند اعمال می نماییم و جهت پیاده سازی این قسمت از قطعه کد زیر استفاده گردیده است:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=16, stratify=y)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
noise_factor = 0.2
X_train_noisy = X_train_scaled + noise_factor *
np.random.normal(loc=0.0, scale=1.0, size=X_train_scaled.shape)
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
```

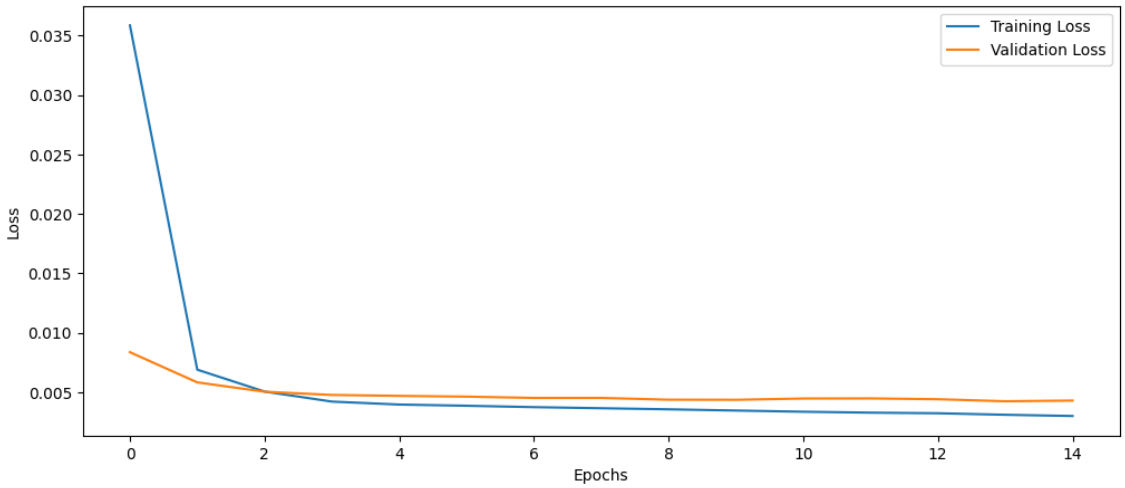
```
X_test_noisy = X_test_scaled + noise_factor *
np.random.normal(loc=0.0, scale=1.0, size=X_test_scaled.shape)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)
input_dim = X_train.shape[1]
encoding_dim = 10
recall_scores1 = []
accuracy_scores1 = []

input_layer = Input(shape=(input_dim,))
classifier1 = Dense(22, activation="relu")(input_layer)
classifier2 = Dense(15, activation="relu")(classifier1)
classifier3 = Dense(10, activation="relu")(classifier2)
classifier4 = Dense(5, activation="relu")(classifier3)
classifier5 = Dense(2, activation="softmax")(classifier4)

classifier_model = Model(inputs=input_layer, outputs=classifier5)
classifier_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model_checkpoint_classifier =
ModelCheckpoint('best_classifier.h5', monitor='val_loss',
save_best_only=True, verbose=1)

history_clf = classifier_model.fit(X_train_noisy, y_train,
                                   epochs=15,
                                   batch_size=512,
                                   shuffle=True,
                                   validation_split=0.2,
                                   callbacks=[
model_checkpoint_classifier])

classifier_model.load_weights('best_classifier.h5')

y_pred = classifier_model.predict(X_test_noisy)
y_pred_classes = np.argmax(y_pred, axis=1)


recall1 = recall_score(y_test, y_pred_classes)
accuracy1 = accuracy_score(y_test, y_pred_classes)

recall_scores1.append(recall1)
accuracy_scores1.append(accuracy1)
rscore_1 = r2_score(y_test, y_pred_classes)
print("R2 Score: ", rscore_1)

plt.figure(figsize=(12, 5))
plt.plot(history_clf.history['loss'], label='Train Loss')
```

```
plt.plot(history_clf.history['val_loss'], label='Validation
Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

cf_matrix = confusion_matrix(y_test, y_pred_classes)

cf_matrix_percent = cf_matrix.astype('float') /
cf_matrix.sum(axis=1)[:, np.newaxis] * 100

plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_percent, annot=True, fmt='.2f',
cmap='Blues', annot_kws={"size": 12})

class_report = classification_report(y_test, y_pred_classes)
print("\nClassification Report:\n", class_report)
```

نتیجه اجرای این برنامه و آموزش با داده های بخش آموزشی نویزی و نامتوازن به صورت زیر گزارش گردیده است:

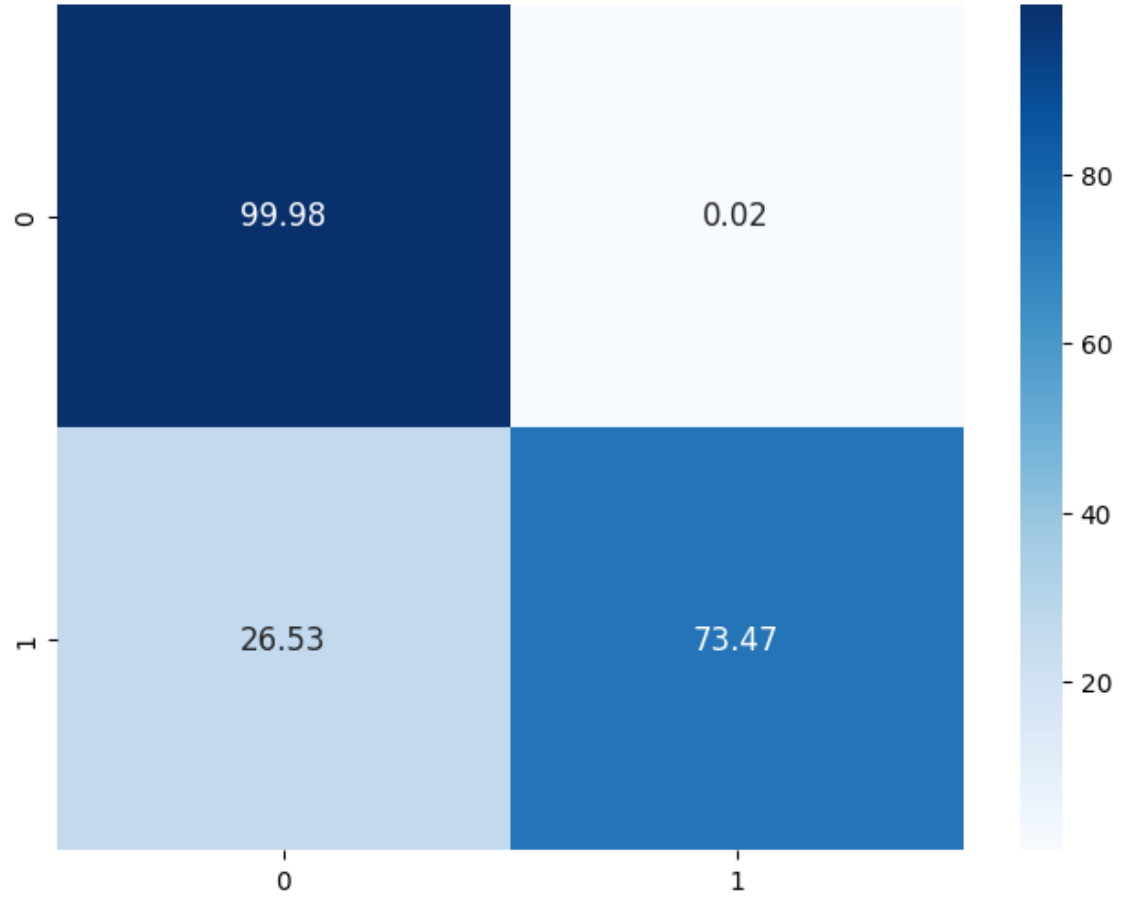
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.84      0.73      0.78        98

    accuracy                           1.00     56962
   macro avg       0.92      0.87      0.89     56962
weighted avg       1.00      1.00      1.00     56962
```

همانطور که انتظار می رفت به دلیل آنکه داده های کلاس تقلب بسیار کمتر از حالت نرمال هستند، Classifier به گونه ای دچار یادگیری بیش از حد کلاس نرمال شده است و با امتیاز f1 یک و دقت ۱۰۰ درصدی تمام داده های این کلاس را تفکیک نموده است درحالی که ۹۸ داده کلاس تقلب را با دقت ۷۸ درصدی و recall ۷۳ درصدی تفکیک نموده است و برخلاف حالتی که از Autoencoder و داده های متوازن گردید، دقت و بازدهی مدل در این حالت نسبت به آن حالت بسیار کم شده است و این ناشی از عدم وجود توازن در کلاس داده ها و نبود اتوانکدر است. نمودار loss function بخش های آزمون و ارزیابی نیز به صورت شکل زیر حاصل شده است:



شکل ۲۸: نمودار loss-function مدل بدون اتوانکدر و با داده های نامتوازن

ماتریس درهمریختگی داده های کلاس ازمون نیز به صورت شکل زیر حاصل شده است:



شکل ۲۹: ماتریس درهمریختگی داده های آزمون بدون اتوانکدر و با داده های نامتوازن

طبق شکل ماتریس درهمریختگی ملاحظه می شود مدل طبقه بند داده های کلاس نرمال یا مشروح را به دلیل آنکه زیاد است خوب فرا گرفته است و با دقتی حدود ۱۰۰ درصد تمامی داده های این کلاس را فرا

گرفته است وازطرفی چون داده های کلاس تقلب بسیار کم نسبت به کلاس مشروح می باشد، مدل این کلاس را به خوبی فرا نگرفته است و با دقتی حدود ۷۳ درصدی داده های این کلاس را تفکیک نموده است.

شایان ذکر است که کد این بخش در این لینک قرار دارد

منابع:

1. https://github.com/ArmanMarzban/MJAHMADEE